Walkowiak

# ATARI® ST

# Grafik und Sound

EIN DATA BECKER BUCH

Walkowiak

# ATARI® ST

# Grafik und Sound

EIN DATA BECKER BUCH

ISBN 3-89011-123-8

Wichtiger Hinweis:
Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

# Vorwort

Grafik und Soundmöglichkeiten - früher die Domäne besonders großer und teurer Spezialrechner - sind dank fortgeschrittener Technik heute auch dem kleinsten und preisgünstigsten Microcomputer zugänglich. So erklärt sich der Erfolg des APPLE II sicherlich zum Teil durch die hochauflösende Grafik (HiRes), die dieser Rechner seinen Anwendern schon 1978 bot. Das große Interesse an Computergrafiken schuf jedoch erst der Commodore 64, war er vom Preis her doch so angesiedelt, daß auch Familienväter, die nur ein neues Hobby suchten und nicht an eine berufliche Nutzung dachten, diesen Heimcomputer erwerben konnten. 320 mal 200 Punkte, auf die die 64er Gemeinde immer besonders stolz war, weckten das Interesse unter anderem auch an professionellen Anwendungen wie CAD. Doch stößt ein System wie der C64 hier an seine Grenzen, sowohl was die rein technischen als auch softwaremäßigen Möglichkeiten angeht.

Richtungsweisend für die 90er Jahre ist das Konzept der Systemfamilie ATARI ST: ein echter sechzehn-Bit-Prozessor, Soundchip, Videocontroller und ein spezielles Datensichtgerät, um die 256000 Punkte auch in hervorragender Qualität darstellen zu können.

Doch was nützt dem stolzen Besitzer eines solchen Computers das Wissen, daß sich unter dem Gehäuse ein Videocontroller oder ein mehrstimmiger Tongenerator wie der AY-3-8910 von General Instruments befinden, wenn er nicht weiß, wie er deren Eigenschaften dazu nutzen kann, um mathematische Funktionen grafisch darzustellen oder sein Keyboard in ein Orgelmanual zu verwandeln?

Zwar geben zu den Hochsprachen mitgelieferte Handbücher Aufschluß über die Befehle, die zur Verfügung stehen, um diese ICs anzusprechen, doch die Algorithmen, also die Lösungsvorschriften für ein bestimmtes Problem, muß der Anwender selbst entwickeln oder entsprechender Literatur entnehmen.

Diesen mühsamen und zeitaufwendigen Prozeß will Ihnen dieses Buch ersparen, es möchte Ihnen ein unentbehrlicher Ratgeber und Lehrmeister sein und soll Ihnen helfen, alle Probleme der Grafik- und Soundprogrammierung zu meistern.

Auch auf Besonderheiten des Gerätes wird dabei eingegangen werden, beispielsweise werde ich Ihnen unter anderem auch zeigen, wie Sie Systemroutinen des GEM in BASIC-Programme einbinden.

Natürlich wird auch DR. Logo nicht zu kurz kommen, schließlich ist diese Sprache für einige Aufgabenstellungen - und für den Einsteiger in Sachen Grafik - besonders geeignet.

Wie schon am Titel des Buches ersichtlich, gliedert sich dieses Buch in zwei Teile. So werden Sie zunächst vom Punkt bis hin zum dreidimensionalem Bild alles über die grafischen Fähigkeiten Ihres ST erfahren. Anhand von zahlreichen Beispielprogrammen werden Ihnen dabei Techniken der Grafikprogrammierung in Theorie und Praxis vorgeführt.

Gleiches gilt für die Programmierung von Tönen, Geräuschen und Melodien, denn diesen Themenkreis behandelt der zweite Teil.

Dabei werde ich Ihnen eine Vielzahl von Programmen vorstellen, die entweder in BASIC, Logo, C oder Modula-2 realisiert worden sind. Bei einigen interessanten Programmen finden Sie diese auch in mehreren Sprachen implementiert - schließlich sollen Sie nicht zur Anschaffung von C gezwungen werden, wenn Ihnen Modula zur Verfügung steht.

Wie dem auch sei, ich nehme an, daß ich mit dieser Vorrede Ihr Interesse geweckt habe, so daß ich zum eigentlichen Thema des Buches kommen kann.

Es bleibt mir nun nur noch zu hoffen, daß Ihnen die Arbeit mit diesem Buch genau soviel Freude bereiten wird, wie mir das

Schreiben und die Entwicklung der hier vorgestellten Programme.

Abschließend möchte ich mich bei allen bedanken, die zum Gelingen dieses Buches beigetragen haben.

Ganz besonders richtet sich mein Dank an Frau Brigitte Witzer, die den Text redigiert hat, und die Herren Helmut Retzlaff, Jürgen Steigers, Thomas A. Vervost und Axel Beier, die wesentlich zum Gelingen des Buches beigetragen haben.

Recklinghausen, Januar 1986 Jörg Walkowiak

[illegible] und die Entwicklung der [illegible] [illegible].

Abschließend möchte ich mich bei allen bedanken, die zum Gelingen dieses Buches beigetragen haben.

Ganz besonders danke ich [illegible] Frau [illegible] Brigitte Witzer, die den Text redigiert hat, und die Herren Helmut [illegible], [illegible], Thomas A. [illegible] und Axel Bauer, die wesentlich zum Gelingen des Buches beigetragen haben.

[illegible], [illegible] 1986 [illegible]

# Inhaltsverzeichnis

**Kapitel 1: Die Grundlagen**



**Kapitel 2: Vom Punkt zum Kunstwerk**

**Kapitel 3: 2-D-Grafik**



**Kapitel 4: 3-D-Grafik**

# 1. Kapitel

# Die Grundlagen

## 1.1 Einführung

Computergrafik ist zur Zeit immer noch das Reizwort im Umgang mit Datenverarbeitungsanlagen.

Egal ob groß oder klein -,womit an dieser Stelle sowohl der Rechner als auch sein Benutzer gemeint sein kann - keiner kommt mehr mit einem Computer aus, der nur rechnen oder Texte verarbeiten kann.

Und so wundert es auch niemanden, wenn die Anbieter der verschiedensten Systeme sich in einem andauernden Wettstreit laufend überbieten: Ein größerer, im Zugriff außerdem schnellerer Speicher, eine höhere Auflösung oder noch mehr Farben lassen den gestern noch gefeierten Micro in Vergessenheit geraten. Allein durch die Ausstattung seines Rechners mit einer benutzerfreundlichen Tastatur kann kein Hersteller seine Marktposition mehr halten. Neue Konzepte, wie beispielsweise die vollständige Integrierung der Maus in das komplette System als Voraussetzung für den Einsatz eines grafisch orientierten Betriebssystemes wie GEM, weisen den Weg in die Zukunft.

Aber was wird durch die neue Technik alles möglich?

Im professionellen Bereich der elektronischen Datenverarbeitung lassen sich zahlreiche Anwendungsbeispiele für Computergrafiken finden; in vielen Aufgabenbereichen ist der Einsatz eines Computers sogar erst in neuester Zeit aufgrund der gesteigerten grafischen Leistungen möglich geworden.

Versehen mit dem richtigen Programm ersetzt ein Großrechner ein ganzes Flugzeug - im Flugsimulator können alle Aspekte des wirklichen Fliegens künstlich erzeugt und realitätsgetreu dargestellt werden.

In der Tat gilt eine Ausbildung am Flugsimulator dem Realflug inzwischen als gleichgestellt, weshalb diese Möglichkeit von den großen Fluggesellschaften genutzt wird, denen es auf diese Weise möglich ist, die Ausbildungskosten beträchtlich zu senken.

Im Bereich der Naturwissenschaften gelten die Computer schon seit langem als unentbehrlich, weil niemand den enormen Berg der anfallenden Meßdaten in vertretbarer Zeit aufarbeiten und natürlich auch kein einzelner Mensch das gesamte Wissen der Menschheit parat haben kann. Und Forschung, wie sie heute betrieben wird und wo es beispielsweise um komplexe Zusammenhänge in der Plasmaphysik oder Genetik geht, wäre ohne Computersysteme mit ihren Datenbanken und Programmen zur visuellen Aufbereitung von Zahlenkolonnen undenkbar.

Denn die menschlichen Sinne reichen schon lange nicht mehr aus, um dem Wissenschaftler bei seiner Arbeit eine ausreichende Hilfe zu sein.

Konnte er allerdings bislang sein Vorstellungsvermögen zu Hilfe nehmen, um sich ein Bild von den Ergebnissen seiner Berechnungen machen zu können, sieht er sich heute unüberwindbaren Problemen gegenüber, wenn es darum geht, komplizierte Erbstrukturen oder Moleküle zu vergleichen.

Hier ist das grafikfähige Terminal, auf dem sich die Moleküle in einer Simulation drehen und von allen Seiten vergrößert und verkleinert betrachten lassen, mit Sicherheit eine unentbehrliche Arbeitshilfe.

Selbstverständlich müssen es nicht unbedingt Atomgruppen sein, die sich mathematisch erfaßt im Speicher des Computers befinden, und deren Bildnisse dem Betrachter auf dem Videoschirm dargeboten werden, häufig sind es auch Autos und Häuser, ja sogar ganze Wohnblöcke. Befindet sich unter den Ausgabegeräten des Computers dann auch noch ein Plotter, kann sich der Architekt bei seiner Arbeit ganz und gar der künstlerischen Seite seiner Entwürfe widmen. Die Routinearbeiten, nämlich sämtliche Berechnungen durchzuführen und den gesamten Komplex maßstabgetreu in sämtlichen Ansichten zu zeichnen, kann dann der Computer während der Pause oder in der Nacht erledigen.

Mit der Unterstützung seines Computers (daher CAD - Computer Aided Design) hat der Architekt quasi im Handumdrehen Arbeiten erledigt, für die er früher Wochen benötigte. Als Beispiel mögen hier die Zeichnungen der Abbildung 1 gelten, die eigentlich auf dem Reißbrett entstanden sein müßten. Tatsächlich wurden sie jedoch mit einem CAD-Programm erstellt, das maßstabsgetreues Zeichnen auf dem Bildschirm und die anschließende Ausgabe der fertigen Skizze auf einem beliebigen Peripheriegerät (Plotter, Drucker) erlaubt.

In der Fertigungsindustrie können die so erarbeiteten Daten dann sogar in entsprechend ausgerüstete Drehbänke (CNC-Maschinen) eingespeist werden, so daß der Mensch seine bildhaften Vorstellungen eines Objektes vollautomatisch realisieren kann. Übrigens wird der gesamte Prozeß dann als CAM, als Computer Aided Manufacturing, bezeichnet.

Aber nicht nur in Forschung und Industrie können diese Supercomputer eingesetzt werden, sondern dank ihrer hervorragenden Grafikeigenschaften finden sie auch immer mehr in künstlerische Bereiche Eingang.

So entstehen in Amerika und Japan Computergrafik-Studios, die mit ihren Arbeiten zeigen, daß selbst die ausgefallensten Objekte überzeugend dargestellt werden können, wobei die Bildqualität in Hinblick auf Schärfe und Farbtreue so hervorragend ist, daß bereits der erste Science-Fiction-Film, der unter dem Titel 'The Works' am New Yorker Institute of Technology entsteht, kurz vor seiner Uraufführung steht.

Wurden bei 'TRON' in den Disney-Studios noch Realfilm und Computergrafik zusammenkopiert, so verwenden die New Yorker Produzenten nur noch synthetische Sequenzen. Es scheint fast so, als wären die Zeiten der Blue-Screen-Technik schon wieder Vergangenheit. Vermutlich werden es in einigen Jahren hochbezahlte Schauspieler sein, die sich vor dem 'Kollegen Computer' fürchten.

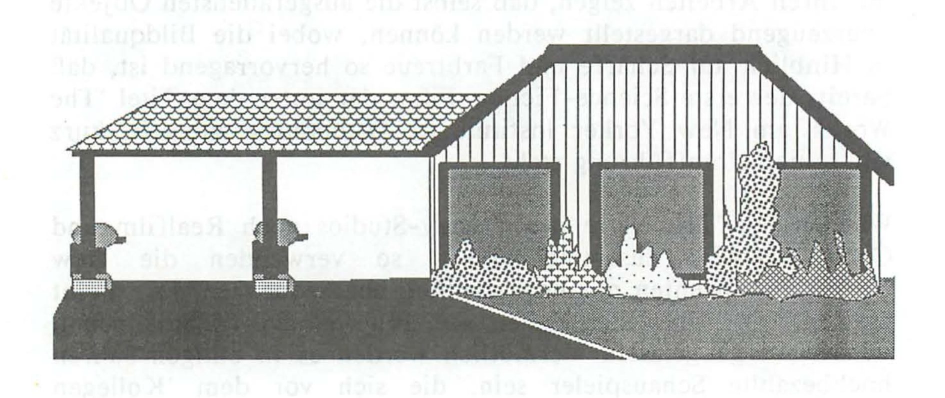


Abb. 1: Beispielausdruck eines CAD-Programmes

Gerade dieses Beispiel zeigt, wie perfekt heutige Computergrafik sein kann, feinste Differenzierungen des Computerbildes sind möglich.

So hat ein Computer-Team der Gesellschaft Lucasfilm, die seit der Star-Wars-Trilogie unbestreitbar zur Elitegruppe in Sachen Bildmontage gehört, unlängst unter dem Titel 'Point Reyes' das Bild einer Landschaft vorgestellt, das jeder unbefangene Betrachter für die Fotografie einer Gebirgslandschaft halten muß. Deutlich sichtbar liegt im Vordergrund eine Straße, kleine Pfützen mit vom Wind gekräuselter Oberfläche und ein Regenbogen beleben das Bild.Und in der Ferne, verhüllt vom Dunst der Atmosphäre, zeichnet sich ein Gebirgsmassiv ab. Und trotz allem handelt es sich um ein synthetisches Produkt, allein erzeugt durch eine ausgeklügelte Software.

Dieses Beispiel macht den Stand der Technik deutlich, den Nutzen für die Allgemeinheit zeigt eher eine Anwendung solcher Grafiksysteme in der Medizin - feinste Gewebeveränderungen macht der Tomograph dem Arzt auf einem Bildschirm sichtbar, und dieser kann dadurch unzähligen Menschen helfen.

All diese Anwendungen wären undenkbar ohne die Möglichkeit, Bilder mit all ihrer Vielfalt an Farben und Formen unter Zuhilfenahme einer entsprechenden technischen Ausrüstung zu erzeugen.

Beispiele haben gezeigt, daß die Grafikfähigkeiten eines Systems unabdingbare Voraussetzung für die verschiedensten Computeranwendungen ist. Und so wird denn auch ständig weitergeforscht und entwickelt, es werden neue Wege gesucht und probiert; der eine Hersteller hat das System des anderen unauffällig kopiert, dabei aber auch perfektioniert, ein anderer präsentiert dafür etwas völlig neues.

Glücklicherweise profitieren davon nicht nur einige ausgesuchte Professoren und die Industrie, sondern es kommt dabei auch etwas für den Anwender von Home- und Personalcomputern heraus.

Denn auch ihm geht es um Punkte und Linien, um Sprites und Shapes, um Farben und um Formen.

Computergrafik ist somit nicht nur ein Schlagwort der Industrie: Kein Heimcomputer ließe sich heute verkaufen, wenn er nicht zumindest in der Lage wäre, Spielegrafiken auf dem heimischen Bildschirm zu erzeugen. Und so ist es kein Wunder, wenn auch die Herstellerfirmen, die den Heimbereich für sich erschlossen haben, ihren Computern einige grundlegende Fähigkeiten mitgeben, die auch auf diesen Microcomputern eindrucksvolle Ergebnisse erzielen lassen.

## 1.2 Die Hardware - Grenze des Möglichen

Die Technik, die sich in unserem ATARI ST befindet, ermöglicht es uns zwar immer noch nicht, real erscheinende Filmsequenzen auf dem Monitor zu erzeugen, bereitet uns aber doch den Weg zu einer Reihe von effekt- und auch sinnvollen Anwendungen.

Wie sich bereits dem vorangegangenen Text entnehmen ließ, erfordern ansprechende Grafiken ein möglichst hohes Auflösungsvermögen wie auch eine Vielzahl von Farbschattierungen. Selbstverständlich ist unser Computer der 3000-DM-Klasse in bezug auf seine Leistung noch weit von den Fähigkeiten einer mehrere Millionen Mark teuren Cray entfernt, die eine Million einzelner Punkte auf einem Grafikterminal darstellen kann, doch verschaffen die immerhin 256.000 einzeln ansprechbaren Punkte, die im Fachjargon übrigens Pixel heißen, dem ST in einer Vergleichsliste der gängigsten Heim- und Personalcomputer wie Apple, Commodore und IBM den ersten Platz.

Es ist für jedermann einzusehen, daß der Computer sich in seinem Speicher merken muß, ob ein einzelnes Element des Bildes nun hell oder dunkel sein soll; ist jedem Punkt auch noch eine bestimmte Farbe zugeordnet, so wird sich dieser Speicherbedarf noch vergrößern. Doch liegt es nicht allein an der Größe des verfügbaren 'Video-RAM' wie wir im folgenden noch sehen werden. Schließlich heißt der ATARI ST in Insiderkreisen auch 'Megajack', und wie Sie vielleicht bereits wissen beansprucht der Bildschirmspeicher des ST von diesen 1000 kBytes nur ca. 32. Unter diesem Gesichtspunkt wären also weitaus mehr als 640 x 400 Punkte machbar gewesen.

Betrachtet man die Modellvielfalt eingehender, so stellt man fest, daß die Hersteller Kompromisse geschlossen haben und die Geräte bzw. deren Betriebsarten sich in vier Gruppen einteilen lassen:

1. Das Gerät verfügt über ein hohes Auflösungsvermögen, zur Darstellung werden allerdings nur wenige (meistens zwei) Farben verwandt.

2. Eine mehrfarbige Darstellung ist nur bei wesentlich niedrigerer Auflösung möglich.

3. Hoch aufgelöste Abbildungen lassen sich farblich fein abgestuft darstellen.

4. Bei allerhöchster Auflösung stehen zahllose Farben zur Verfügung.

Punkt eins und zwei kamen im Atari ST zur Geltung, und es bleibt zu sagen, daß die Entwickler des ST hier bei der Auflösung von 640 x 400 Punkten bei monochromer Darstellung und 320 x 200 Punkten bei sechzehn gleichzeitig dargestellten Farben einen guten Kompromiß geschlossen haben. Wobei zu beachten ist, daß diese sechzehn Farben aus einem Satz von 512 gewählt werden können!

Punkt drei und vier wären zwar technich ebenfalls in einem Rechner mit Heimcomputergröße machbar, doch müßte die Hardware - und damit auch der Preis - dann ein wenig anders aussehen, wie im folgenden kurz gezeigt werden soll.

### 1.2.1 Rastergrafik

Bei den von uns angewandten Systemen handelt es sich, sofern wir nicht mit einem speziellen Grafiksystem an irgendwelchen Institutionen arbeiten, durchweg um Rastergeräte. Um von diesem Verfahren zur Darstellung beliebiger Bilder eine Vorstellung zu bekommen, stellen Sie sich bitte vor, daß Sie ein feinmaschiges Gitternetz über das Bild legen. Haben Sie dazu ein Fliegengitter benutzt, entsteht ein Bild niedriger Auflösung, bei Verwendung eines feinmaschigen Damenstrumpfes hätten Sie ein hohes Auflösungsvermögen erzielt. Betrachten Sie dann das so 'gerasterte' Bild, können Sie zahllose kleine Bildflächen erkennen - und auch explizit angeben. So könnte ein Bild einer vorgegebenen und konstanten Größe aus vielleicht 400 Bildzeilen bestehen, von denen jede wiederum 640 Bildpunkte erhält.

Als nächstes legen wir dann fest, daß es nur die beiden Farben schwarz und weiß geben soll, und bestimmen dann für jedes einzelne Feld unseres Rasters anhand dessen Helligkeit, ob es sich um einen weißen oder schwarzen Punkt handeln soll. Summa summarum werden wir 256.000 einzelne Punkte betrachten müssen. Notieren wir dann für jeden schwarzen Punkt eine 1 und für jeden weißen Punkt eine 0, dann haben wir das Bild erfolgreich digitalisiert.

Und genau so steht das Bild dann auch im Bildschirmspeicher eines Computers, der sich der Rastergrafik bedient. Nur arbeitet dieser nicht mit einzelnen Punkten, sondern vielmehr mit Bytes (Achtergruppen von Bits), d.h. wir benötigten zur Speicherung obigen Bildes exakt 32.000 Bytes, also 31,25 kByte.

Um nun ein Bild höherer Auflösung, sagen wir von vielleicht 1024 x 1024 Punkten, darstellen zu können, müßten wir schon

ca. 130 kByte aufwenden. Dies ist jedoch nicht einmal das Problem - zumindest nicht in der heutigen Zeit, wo der Mega-Chip zur Serienreife gelangt und die Preise fallen.

Problematisch ist vielmehr der Zeitaufwand, der benötigt wird, um solch ein Bild auf dem Monitor darzustellen. Denn bei Rastergeräten wird das Bild Zeile für Zeile aufgebaut, der Elektronenstrahl huscht also 640 mal von links nach rechts, um ein Bild zu erzeugen. Um jedoch ein Flimmern des Bildes zu verhindern, geschieht das ganze mindestens 25 bis 30 mal pro Sekunde (beim ST mit angeschlossenem Monochrom-Monitor sogar 70 mal), erst dann erweckt die Trägheit unserer Netzhaut das Gefühl eines stehenden Bildes.

Für einen Computer bedeutet das nicht mehr und nicht weniger, als daß 130 000 Speicherplätze dreißig mal in der Sekunde abgefragt werden müssen. Außerdem will der Prozessor auch während der Abarbeitung eines Programmes auf den Bildspeicher zugreifen, das Bild muß schließlich immer irgendwie manipuliert werden. Rechnen Sie sich nun einmal aus, wie viele Leseoperationen innerhalb einer Sekunde durchgeführt werden müßten, um eine Grafik von 4096 x 4096 Punkten abzubilden!

### 1.2.2 Vektorgrafik

Bei der Vektorgrafik geht man nun einen ganz anderen Weg. Hier wird der Elektronenstrahl nicht mehr waagrecht mit einer konstanten Wiederholfrequenz über den Schirm gejagt, sondern er wird gezielt gesteuert. Wo nichts abzubilden ist, da kommt er auch nicht hin, sondern er steuert nur die wirklich gesetzten Positionen an. Im allgemeinen wird dieses Verfahren Zeit einsparen, da niemals alle Punkte der Bildfläche angesteuert werden müssen (was bei Rastergrafik immer der Fall ist).

### 1.2.3 Der Bildschirmspeicher des ATARI ST

Der Atari läßt sich eindeutig der ersten Gruppe zuordnen, und so fragen wir uns zunächst einmal, welchen Bereich seines Speichers er für die auf dem Monitor erscheinende Abbildung bereithält. Nun, sofern der Bildschirmspeicher im normalen Adreßbereich und nicht in einer Extra-Bank liegt, sollte man das, selbst wenn man keinerlei Unterlagen hat, durch gezieltes Schreiben in die verschiedensten Speicherstellen herausfinden können. Bei den 260 und 520 ST's zeigt sich eine erste Reaktion beim Beschreiben der Adresse &H78000 (dez. 491519); der Benutzer eines ST+ wird sein Erfolgserlebnis hingegen erst bei Speicherzelle &HF8000 (dez. 1015808) haben:

```
POKE 491520,65535
POKE 491520,0
```

Während der erste Befehl einen deutlichen Strich zeichnet, löscht der zweite Befehl alle gesetzten Punkte wieder.

Wer bereits mit bit-mapped-Grafiken gearbeitet hat, der fragt sich nun, warum 65535, obwohl doch die Dezimalzahl 255 (bzw. &HFF) den acht gesetzten Bits eines Bytes entspricht. Doch derjenige darf nicht vergessen, daß er mit dem Umstieg zum Atari einen echten Sprung getan hat: des Rätsels Lösung liegt nämlich in der Natur des 68000-Prozessors begründet.
Zwar kann der Prozessor auch Bytes adressieren, Poke ist jedoch als sechzehn-Bit-Befehl ausgelegt.

Eine Zeile unseres Schirmes besteht aus 80 Bytes (640 Punkte durch 8 Bits), somit müßten 40 Pokes (da immer gleich sechzehn Bit gesetzt werden) reichen, um die Zeile zu füllen:

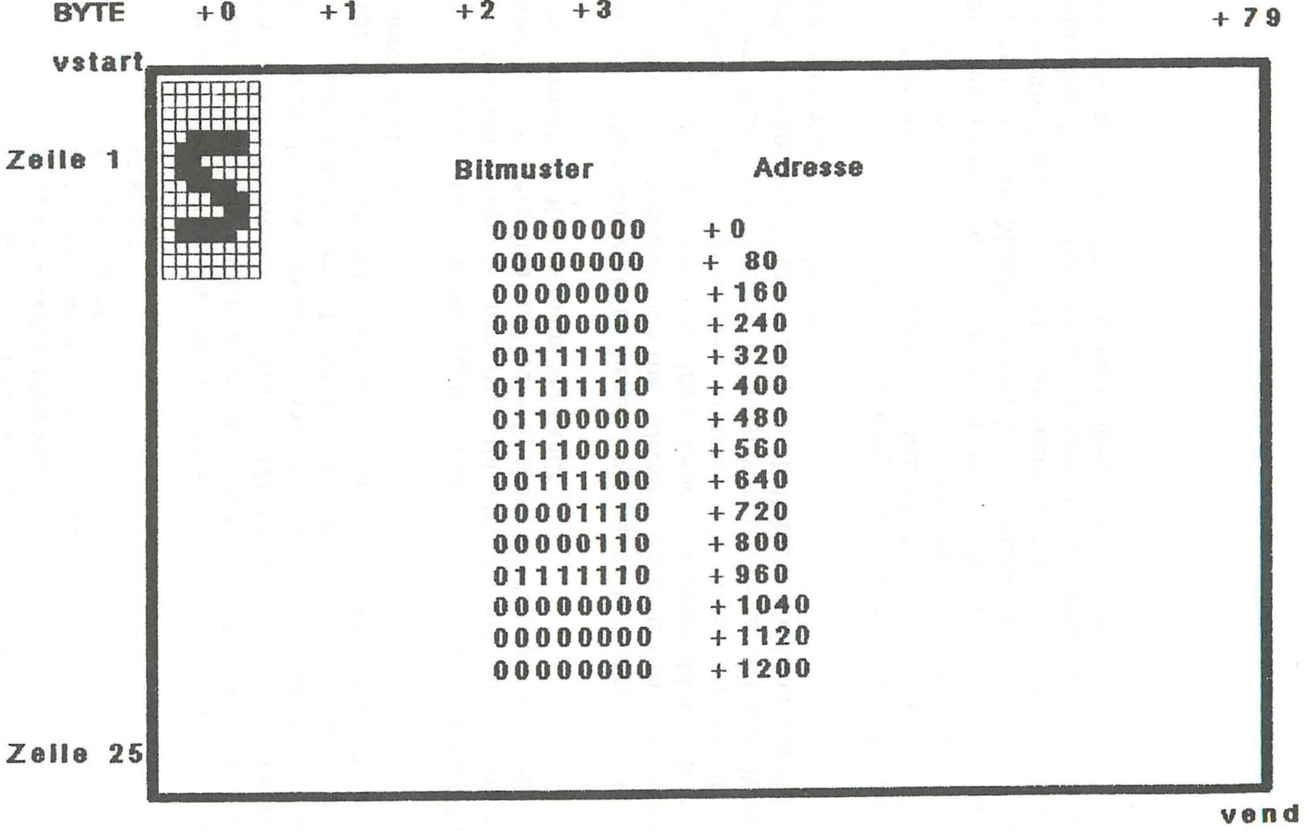



Abb. 2: Bildschirmorganisation bei HiRes

```
5 REM Die Zahlen in Klammern gelten fuer ST+
10 VSTART = 491520 (1015808)
20 FOR I=VSTART TO VSTART + 79 STEP 2
30 POKE I,65535
40 NEXT I
```

Diese Zeilen beweisen die Richtigkeit unserer Überlegungen. Rein rechnerisch müßte unser Bildschirm-RAM somit bis einschließlich zur Adresse 523519 (1047806) gehen, was ein kurzer Test auch beweist: POKE 523519,0 löscht die unterste rechte Ecke auf dem Schirm. Diese Beziehungen zwischen Punkt und Adresse wie auch Adressen und Bild macht Abbildung 2 noch einmal deutlich.

Dieser Skizze können Sie auch entnehmen, wie der ATARI ST die Zeichen seines Zeichensatzes abbildet. Jeweils 16 der insgesamt 640 Rasterzeilen und acht Spalten werden zur Darstellung eines einzelnen Buchstabens verwandt.

Die einzelnen Rasterzeilen liegen, wie zuvor bewiesen, 80 Bytes auseinander - weshalb sich die Adressen, an denen die Bitmuster zur Darstellung einer einzelnen Zeile des abgebildeten Buchstabens 's' abgelegt werden müssen, wie angegeben ermitteln lassen. Wenn wir wollten, könnten wir auf diese Art und Weise jedes gewünschte Zeichen an jeder beliebigen Position des Bildschirmes erscheinen lassen.

Wir müssen dabei nur beachten, daß immer gleich zwei Speicherstellen angesprochen werden. Deshalb sollten Sie es sich zur Gewohnheit machen, immer nur die geraden Adressen anzusprechen (was sich vor allem dann bezahlt macht, wenn Sie später einmal in Maschinensprache programmieren wollen). Mit ein wenig Umdenken sollten Sie dann in der Lage sein, jede gewünschte Pixelposition anzusprechen. Ungewohnt werden dabei zunächst nur die auftretenden großen Zahlen sein:

```
. . . . . . . . . . . . . . . .  = 0

. . . . . . . . . . . . . . . *  = 1

. . . . . . . . . . . . . . * .  = 2

. . . . . . . . . . . . . . * *  = 3

. . . . . . . . * * * * * * * *  = 32767

* * * * * * * * . . . . . . . .  = 65280

* * * * * * * * . . . . . . . *  = 65281

* * * * * * * * * * * * * * * *  = 65535
```

Wie Sie sehen, werden Sie Werte zwischen eins und &HFFFF 'poken' müssen, um innerhalb eines Wortes die gewünschte Bitkombination setzen zu können.

Außerdem sollte noch erwähnt werden, daß das Bildschirm-RAM nicht immer auf diesen Adressen liegen muß. Während des Bootens sorgt daß Betriebssystem jedoch dafür, daß zunächst immer die letzten 32 kByte des wirklich vorhandenen Speichers reserviert werden. (Abgesehen von einigen Bytes, die die MMU im Bereich zwischen FF 0000 bis FF FFFF als I/O-Bereich für die Ansteuerung der Peripherie belegt.)

Nicht ganz so einfach, aber immer noch logisch und konsequent, sieht die ganze Sache aus, wenn ein Farbmonitor angeschlossen wurde. Das fehlende Signal 'Monochrome Monitor Detect', welches vom SW-Monitor ausgesandt wird, veranlaßt das Betriebssystem, den Farbmodus einzuschalten. Das Auflösungsvermögen bleibt auf 320 x 400 (bzw. 320 x 200) Punkte beschränkt, dafür funktionieren aber die Schieber innerhalb des Control Panels. Die Einstellung geht denkbar einfach vonstatten:

Wählen Sie in den untersten beiden Zeilen einen Zeichenstift aus, und mischen Sie sich aus den Grundfarben rot, grün und blau einen Ihnen zusagenden Farbton zusammen.

Auf sechzehn Stifte kann der Atari zugreifen, wenn die Low-Resolution-Betriebsart eingeschaltet ist, sechzehn verschiedene Zustände kann somit jeder Bildschirmpunkt annehmen. Zwei hoch vier Bits eines Bytes im Video-RAM fallen somit der Farbdarstellung eines jeden Punktes zum Opfer. Und da wir den Bildschirmspeicher nicht beliebig vergrößern können, dieser auf seine 32 kByte beschränkt bleibt, bleiben effektiv somit nur noch 64.000 Bits für die reinen Punkte übrig.

Ein Byte enthält nicht mehr acht Punkte in einer Farbe, sondern zwei Punkte in sechzehn Farben (2 x 4 Bit). Wie der ST seinen Bildschirmspeicher organisiert, wenn eine der beiden Farbbetriebsarten eingestellt wurde, machen Abb. 3 und Abb. 4 deutlich.

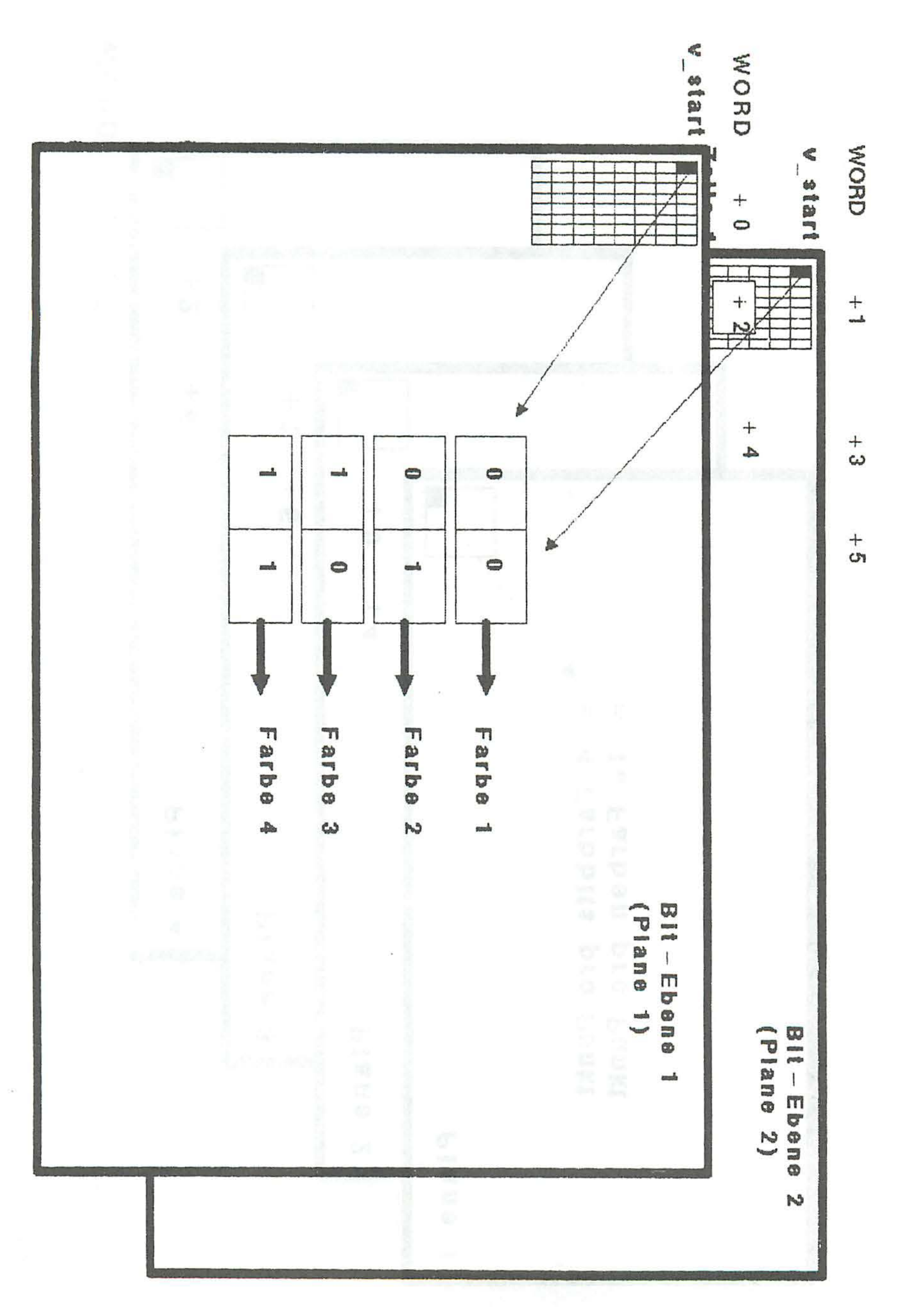



Abb. 3: Bildschirmorganisation bei MidRes

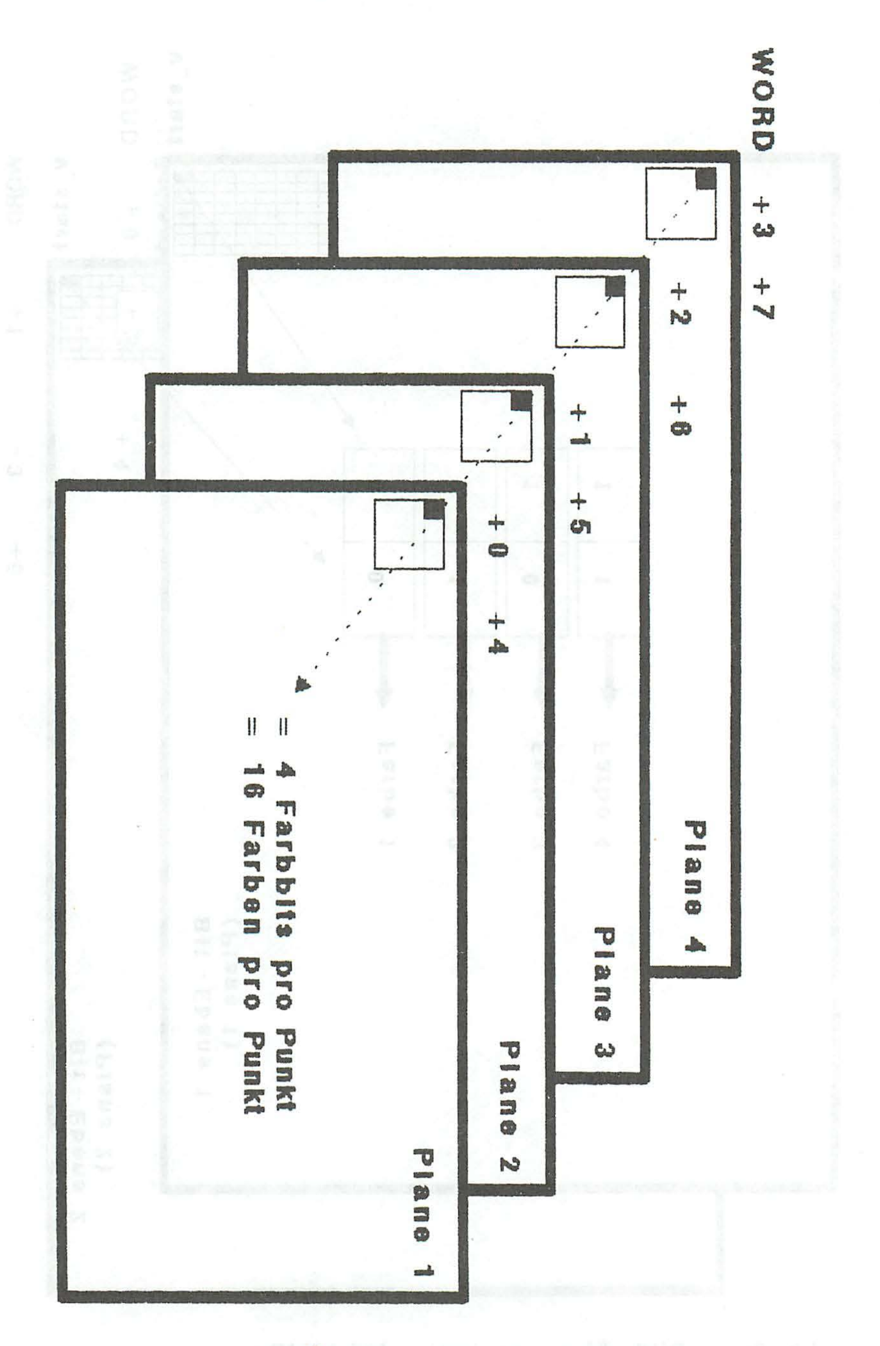



Abb. 4: Bildschirmorganisation bei LowRes

## 1.3 Die Betriebssoftware - Unterstützung des Machbaren

Wohl jeder von uns hat bereits mit den unterschiedlichsten Mitteln versucht, sich künstlerisch zu betätigen, sei es nun im Kindergarten mit Wand- oder Wasserfarbe oder in der Schule mit Bleistift und Tuschefüller, oder aber auch zu Hause mit Glas- beziehungsweise Ölfarbe.

Alle Techniken hatten eines gemeinsam: So wie der Maler zunächst erst seine Staffelei aufbaut, so haben auch wir uns erst die Arbeitsmaterialien besorgt, haben den Bildträger gewählt und uns unsere Lieblingsfarben aus dem erhältlichen Angebot ausgesucht, dann erst, nach einer Phase der Besinnung, haben wir mit der eigentlichen Arbeit begonnen.

Unser Atari stellt uns nun vor das Problem der Wahl gleich in mehrfacher Hinsicht, müssen wir uns doch zunächst erst einmal entscheiden, welche Sprache zum Einsatz gelangen soll. BASIC, Logo, C oder Assembler?

Vor- und Nachteile bietet eine jede dieser Sprachen. BASIC ist seit Jahren Standard und wird von vielen bereits beherrscht; Grafikprogramme geschrieben in Logo sind dank der Konzeption der Schildkröte recht verständlich, da in die reale Begriffswelt übertragbar. C bedeutet Geschwindigkeit und wird außerdem von professionellen Softwareentwicklern verwandt werden, auch deshalb, weil die generierten Maschinencode-Programme sehr schwer durchschaubar sind und deswegen bereits einen gewissen Programmschutz darstellen. Gleiches gilt in besonderem Maße natürlich für Assemblerprogramme, die wohl nur da ihre Berechtigung finden, wo es auf extreme Arbeitsgeschwindigkeit ankommt. Das Argument des geringeren Speicherplatzbedarfs kann beim ST vernachlässigt werden.

Weiterhin ist natürlich die vorhandene technische Ausstattung wichtig für eine anwendungsgemäße Repräsentation. So wird ein jeder für den Umgang mit einer Textverarbeitung oder einer Tabellenkalkulation wohl den hochauflösenden Schwarzweißmonitor vorziehen; sobald man sich jedoch mit Computerkunst

befaßt oder ein Arcade-Game geladen hat, möchte man die Farbe sicher nicht missen.

### 1.3.1 Die Utensilien

Im Grunde genommen wäre es völlig ausreichend, wenn ein Computer nur über eine Anweisung zum Setzen der einzelnen Grafikpunkte auf dem Bildschirm verfügen würde. Und selbst darauf könnte man noch verzichten, wie das vorangegangene Kapitel gezeigt hat. Doch wäre die Programmierung komplexer Grafikanwendungen dann so mühselig, daß man wohl doch lieber wieder zu Papier und Bleistift greifen würde.

Also schufen die Softwareingenieure Unterroutinen, die gleich die Darstellung einer Linie, eines Kreises oder einer Figur übernahmen. Ergänzt wurde das ganze dann um Anweisungen zur Handhabung der darstellbaren Farben. Und so kennt unser ST auch Anweisungen wie PEN und COLOR. Eine Aufstellung und Gegenüberstellung der wesentlichsten Statements finden Sie innerhalb des Kapitels 'Tricks, Tips & noch mehr Grafik'.

### 1.3.2 ... und das Papier des Computerkünstlers

Zur computermäßigen Darstellung des Papieres lassen sich zwei voneinander verschiedene Wege beschreiten. So kann die Zeichenfläche einerseits beliebig groß sein und nur ein Ausschnitt wird auf dem Bildschirm angezeigt (Logo), oder eine Bildfläche konstanter Größe wird gerastert, so daß jede einzelne Position durch Koordinaten ansprechbar ist (BASIC). Natürlich kann auch im ersten Fall ein Bezug zu einem Koordinatennetz hergestellt werden, allerdings ist die Handhabung wegen der fehlenden Grenzwerte erschwert. Daher arbeiten solche Systeme in den seltensten Fällen auch koordinatenbezogen, vielmehr bezieht der Anwender sich auf Vektoren und gibt immer nur relative Distanzen (und Richtungen) zu einem Bezugspunkt an.

Dieser Bezugspunkt ist bei Logo der jeweilige Standort der Schildkröte, die normalerweise immer auf dem Bildschirm sicht-

bar ist. Der Vorteil einer solchen Arbeitsweise liegt auf der Hand, ist die Grafikprogrammierung doch besonders einfach. Denn schließlich müssen keine Koordinaten angegeben oder zuvor gar noch transformiert werden, sondern eine einfache Eingabe wie 'gehe 40 Längeneinheiten vor' oder 'drehe dich um ein viertel nach links' - wobei 'viertel' gleichbedeutend mit 90 Grad ist - ist völlig ausreichend.

# 2. Kapitel

# Vom Punkt zum Kunstwerk

## 2.1 Hinweise zu den Listings

Die Listings in diesem Buch wurden alle mittels eines direkt an den ST angeschlossenen Laserdruckers erstellt. Es handelt sich bei ihnen also um Ausdrucke der lauffähigen Programme, die dann fotomechanisch kopiert wurden, um mögliche Fehlerquellen bei der Übertragung oder Bearbeitung auszuschließen.

Beachten Sie bitte auch, daß einige der vorgestellten Routinen in den verschiedensten Programmen wieder auftauchen und Sie sich ein erneutes Eingeben der betreffenden Programmteile ersparen können. Aus diesem Grunde sollten Sie, soweit es BASIC betrifft, darauf verzichten, die RENUMBER-Funktion einzusetzen. Denn nur dann haben Sie die Möglichkeit, diese Programmteile mit MERGE zusammenfügen.

Soweit es um die Eingabe von Modula-2- bzw. C-Programmen geht, möchte ich Ihnen unbedingt den Gebrauch des Editors der Firma Metacomco nahelegen. - Nicht etwa deshalb, weil ich einen Werbevertrag mit diesem Softwarehaus abgeschlossen habe, sondern weil dieser als einziger zeilenorientiert arbeitet. Sollte Ihnen dieses Programm nicht zur Verfügung stehen, benutzen Sie bitte 1.ST-Word, alle anderen Editoren verhielten sich während der Arbeit zu diesem Buch nicht gerade betriebssicher (Stand: Januar 1986).

## 2.2 Als Einstieg: Grafik mit Logo

Bevor wir uns in den folgenden Kapiteln dieses Buches richtig mit der Computergrafik befassen und um Probleme wie Drehungen, Spiegelungen oder auch Skalierungen kümmern, sollen anhand kurzer Beispiele einige grundlegende Dinge zum Thema Computergrafik erläutert werden. Möglicherweise werden Ihnen einige der folgenden Zeilen dabei bekannt vorkommen, dennoch sollten Sie dieses Kapitel nicht überblättern.

Denn sehr schnell werden wir von simplen zu komplexen Programmen kommen und dabei Details herausstellen, die sich als grundlegend für die späteren Arbeiten erweisen.

### 2.2.1 Logo und die Schildkröte

Doch zunächst zurück zu den Anfängen. Als Seymor Pappert, der Entwickler von Logo, Mitte der siebziger Jahre durch seine erfolgreichen Versuche, Kindern den Umgang mit Computern beizubringen, das Interesse der Weltöffentlichkeit auf sich zog, ging er zu Recht davon aus, das die elektronische Repräsentation eines greifbaren Gegenstandes den Umgang mit Computern stark vereinfachen würde.

Bei seinen Experimenten stellte er ein Kind in die Mitte des Raumes, und die übrigen Teilnehmer an diesem ehrgeizigen Projekt sollten ihren Kameraden nun durch Zurufe derart dirigieren, daß er auf seiner Wanderung mathematische Figuren und Formen abschritt.

Ein Quadrat sah dann folgendermaßen aus:

```
gehe fünf Schritte vor
drehe dich um 90 Grad nach rechts
gehe fünf Schritte vor
drehe dich um 90 Grad nach rechts
gehe fünf Schritte vor
drehe dich um 90 Grad nach rechts
gehe fünf Schritte vor
```

Benutzte die Versuchsperson dabei ein Stück Kreide, um den zurückgelegten Weg zu markieren, so ließ sich das Ergebnis des Versuchs mit den Kindern diskutieren, weitere 'Spiele' dieser Art konnten folgen. So lernten die Kinder auf eine ganz neue Art die Geometrie kennen.

Im nächsten Schritt wurde der Raum dann durch den Bildschirm eines Computers und die Versuchsperson, welche die Zeichnung erstellte, durch eine grafisch dargestellte Schildkröte ersetzt.

Dadurch war es den Kindern möglich, Handlungen in ihrer Realwelt auf dem Computer nachzuvollziehen - und da Kinder

sowieso gerne malen, war der Computer zunächst nichts Anderes als ein komfortables Zeichengerät.

Das Bild des Quadrates aus dem Übungsraum konnte von jedem Mitglied der Übungsgruppe sogleich auf dem Monitor erstellt werden:

```
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
```

Auch Sie sollten nun dieses kurze 'Programm' auf Ihrem ATARI eingeben, kann es doch bereits Grundlage zu ansprechenden Grafiken sein. Zweckmäßigerweise sollten Sie dann aber ruhig die Abkürzungen benutzen: forward = fd, right = rt.

Nennen Sie diese Prozedur Quadrat, und rufen Sie sie anschließend auf. Wie Sie sehen, hat die Schildkröte wieder ihre Grundstellung eingenommen, die Sie jederzeit mit HOME erreichen können.

Um das Abbild des Quadrates nun um einen beliebigen Winkel zu drehen, müssen Sie nur die Turtle entsprechend ausrichten, beispielsweise durch Eingabe von rt 10. Ein erneuter Aufruf von QUADRAT führt dann zum gewünschten Ziel.

Sofern Sie diesen Vorgang dann einige Male wiederholen, haben Sie bereits Ihre erste Computergrafik geschaffen (vgl. Abb. 5).

Sinnvollerweise werden wir solche Manipulationen dann aber nicht von Hand vornehmen, sondern schreiben ein kurzes Programm (das QUADRAT als Unterprogramm aufruft):

```
TO GRAFIK1
QUADRAT
RT 5
GRAFIK1
END
```

Sollten Sie einmal Zeit und Muße haben, schreiben Sie doch einmal eine entsprechende BASIC-Version; Sie werden sehen, daß dort ein mehrfaches an Aufwand getrieben werden muß. Hier hingegen reichen ein paar Zeilen, und mit einigen kleinen Änderungen am Programm lassen sich sogar ganz andere Grafiken erzeugen. So steht es Ihnen frei, Kantenlänge oder Drehwinkel während des Ablaufes zu ändern, oder drehen Sie einmal andere Figuren (beispielsweise Dreiecke oder gar Ziffern und Buchstaben).

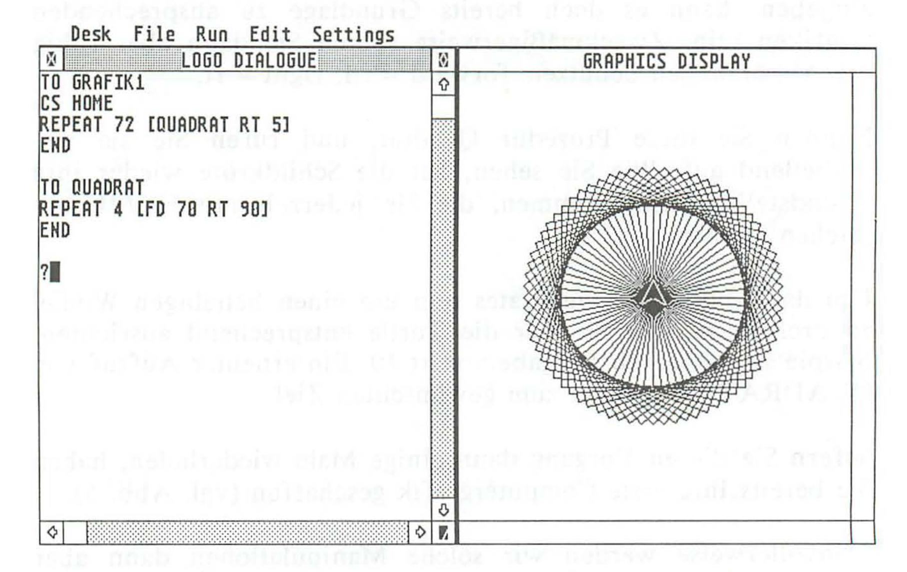



Abb. 5: Grafikprogramme in Logo - kurz aber effektvoll

### 2.2.2 Grafiken zum Anschauen

Im folgenden wollen wir uns mit dem Bereich der Computergrafik befassen, dem am ehesten wohl der Name Computerkunst zusteht, und einige Aspekte kennenlernen, die bei wohl bei jeder Computergrafik Anwendung findet, deren einziges Ziel darin besteht, ästhetisch schön zu wirken.

Ein weiteres Verfahren, das zur Kreation der verschiedensten Computergrafiken eingesetzt wird, vergrößert (verkleinert) die Teilfigur, aus der das Gesamtbild sich zusammensetzt.

Auf diese Art und Weise entstehen dann sinnverwirrende Grafiken, die unterschiedliche räumliche Interpretationen zulassen. So könnten sich Betrachter der Darstellung von Abb. 6 darüber streiten, ob der Quader nun ins Bild hinein- oder aus ihm herausragt.

Bei Anwendung beider Techniken lassen sich dann schon ganz ansehnliche Werke schaffen, die ihren besonderen Reiz durch ihre Tiefenwirkung erhalten (vgl. dazu Programm GRAFIK2 der Abb. 7). Eine geschickte Wahl der Parameter erlaubt im Extremfall sogar eine reelle Deutung:

So macht das Programm GRAFIK3 immer noch nichts anderes, als eine Reihe von Quadraten zu zeichnen. Dennoch unterscheiden die Ergebnisse sich wesentlich von den bislang erzeugten Bildern.

Bildbestimmend ist einzig und allein eine geschickte Wahl der Parameter Seite, Distanz und Drehung. Womit die Seitenlänge des ersten Quadrates, die Distanz der abgebildeten Quadrate zueinander als auch deren Drehwinkel gemeint sind:

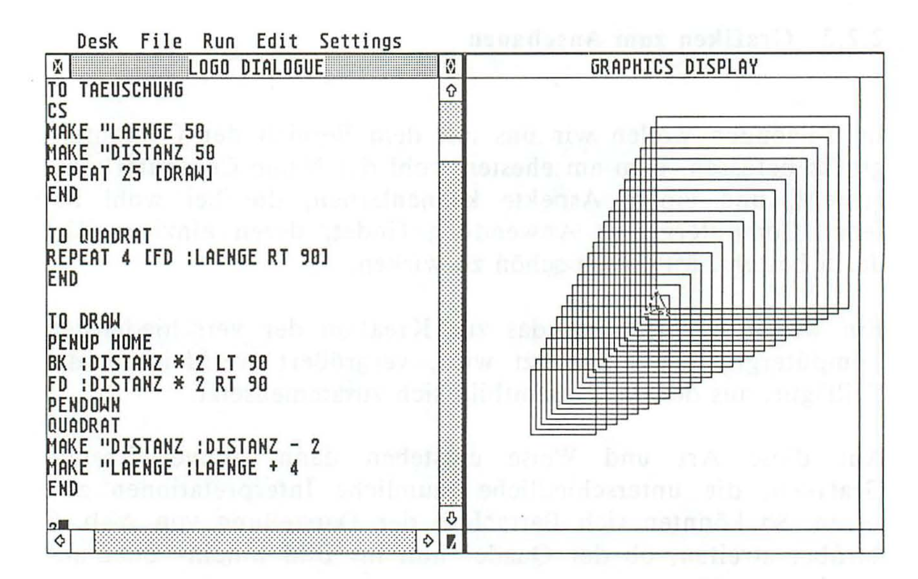



Abb. 6: Das Logo-Programm TAEUSCHUNG

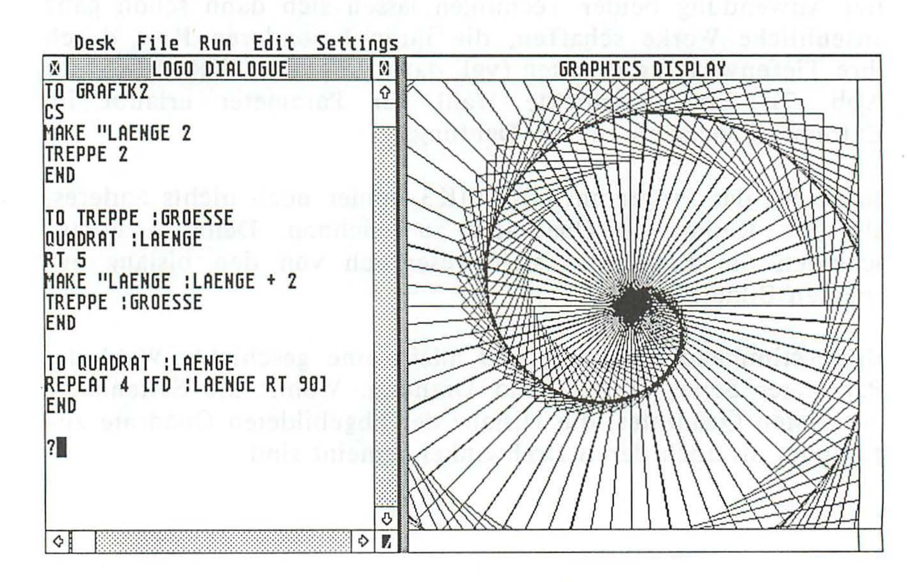



Abb. 7: Das Logo-Programm GRAFIK2

```
TO QUADRAT
REPEAT 4 [FD :LAENGE RT 90]
END

TO START
PENUP HOME
BK :DISTANZ * 2 LT :WINKEL - :DREHUNG
FD :DISTANZ * 2 RT :WINKEL + :DREHUNG
MAKE "WINKEL :WINKEL - :DREHUNG
PENDOWN
QUADRAT
MAKE "DISTANZ :DISTANZ - 2
MAKE "LAENGE :LAENGE + :SEITE
START
END

TO GRAFIK3 :SEITE :DISTANZ :DREHUNG
CS FS
MAKE "WINKEL 90
MAKE "LAENGE 0
START
END
```

Wer würde die durch den Aufruf

GRAFIK3 5 -20 240

geschaffene Grafik nicht als Darstellung von Hochhäusern aus der Vogelperspektive bezeichnen?

Probieren Sie auch:

```
GRAFIK3 20 -20 120
GRAFIK3 -20 20 240
GRAFIK3 2 2 2
GRAFIK3 1 50 240
GRAFIK3 -2 0 10
```

Um nach diesem Prinzip nun eine Vielzahl von Grafiken erzeugen zu können, müßten wir die Routine noch universeller gestalten. Dazu zählt neben einer variablen Eingabe von Größe der Grundfigur, Vergrößerungsfaktor und Drehwinkel auch die Anzahl der Ecken.

Logo macht auch hier die Sache einfach. Wenn man bedenkt, daß ein Kreis üblicherweise aus 360 Teilstrecken besteht, die jeweils um ein Grad versetzt werden, somit also ein 360-Eck ist, dann läßt sich der Winkel zwischen zwei Seiten eines beliebigen Vielecks mit 360 Grad dividiert durch die Anzahl der Ecken angeben.

```
TO DREIECK
REPEAT 3 (FD 100 RT 360/3)
END
```

Diese Prozedur wird die Turtle veranlassen, ein gleichschenkliges Dreieck der Seitenlänge 100 zu zeichnen; beliebige Vielecke lassen sich entspechend der Prozedur VIELECK im Programm GRAFIK4 erstellen. Angemerkt sei hier noch, daß bei einem regelmäßigen Vieleck der Winkel zwischen zwei Seiten immer von konstanter Größe sein wird, weshalb er auch nur ein einziges Mal berechnet werden muß - was natürlich der Arbeitsgeschwindigkeit des Programmes zugute kommt. Ein Trick, den Sie sich merken sollten!

```
TO GRAFIK4 :ECKEN :GROESSE :FAKTOR :WINKEL
CS HT
START :GROESSE :FAKTOR :WINKEL
END

TO START :GROESSE :FAKTOR :WINKEL
VIELECK :ECKEN :GROESSE
FD 2
RT :WINKEL
MAKE "GROESSE :GROESSE + :FAKTOR
START :GROESSE :FAKTOR :WINKEL
END

TO VIELECK :ECKEN :SEITE
LOCAL "WINKEL
MAKE "WINKEL (360 / :ECKEN)
REPEAT :ECKEN [FD :SEITE RT :WINKEL]
END

MAKE "GFILL "TRUE
```

Die Abbildungen auf den folgenden Seiten sind dann durch Aufruf dieses Programmes mit den Parametern

8 60 0 124

und

100 5 0 10

entstanden.

Auch hier sollten Sie wieder mit den verschiedensten Werten experimentieren.

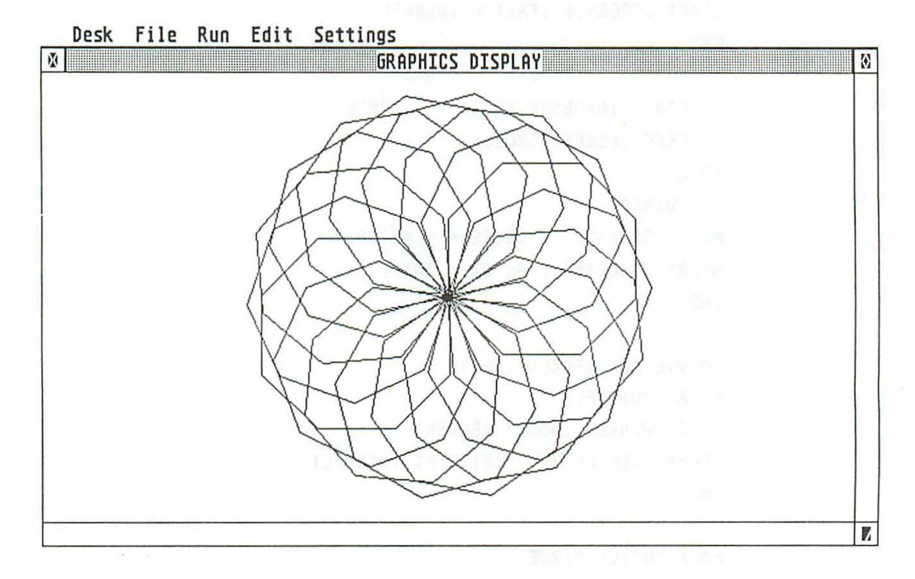



Abb. 8: Hier wurde GRAFIK4 mit 8 60 0 100 aufgerufen

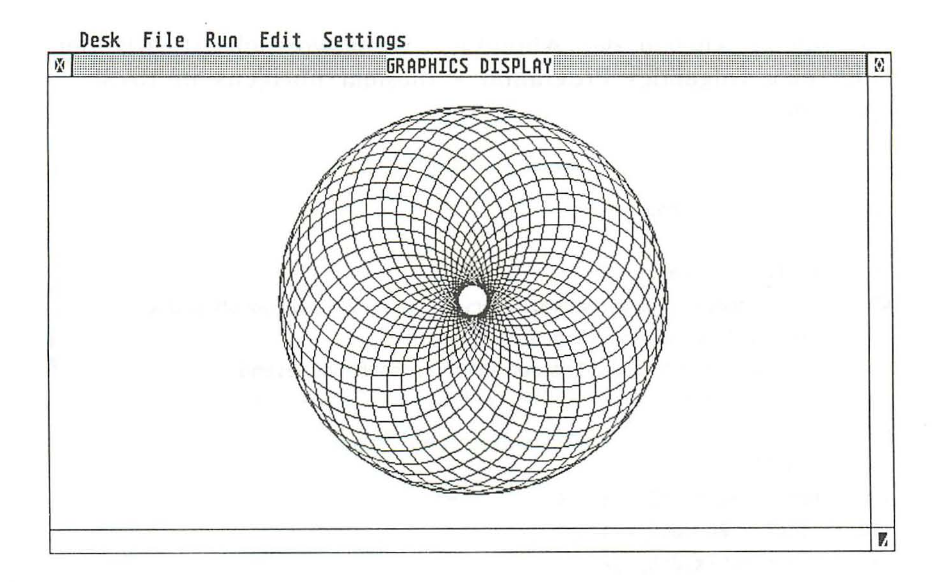



Abb. 9: Ein Bildschirmausdruck von GRAFIK4 100 5 0 10

### 2.2.2.1 Moire, Symmetrie und Rekursion

Wie Sie sehen, lassen sich mit solch einem - obwohl recht kurzen - Programm schon die unterschiedlichsten Grafiken erzeugen.

Was sind jedoch die Tricks, die der Programmierer anwenden muß, um zu solchen Kunstwerken zu gelangen?

Betrachten wir die Bilder eingehender, so stellen wir fest, daß es sich durchweg um symmetrische oder aber um selbstidentische Abbildungen handelt - womit der Weg für den Programmierer ganz klar vorgezeichnet ist.

Denn im ersteren Fall muß dieser einfach eine Folge von Punkten auswählen und beispielsweise von diesen Punkten ausgehend zum Bildrand führende Linien zeichnen, wobei er durch eine einfache Rechnung gleich mehrere Randpunkte ermittelt (z. B. jeweils ein Paar nach der Vorschrift (x, y-m) und (x, y+m) und dadurch zu einer Spiegelachse gelangt, die für den sym-

metrischen Aufbau der Abbildung verantwortlich ist. Als Beispiel mag folgendes Programm - diesmal übrigens in BASIC - genügen:

```
10    rem grafik1.bas
20    rem ----------
30    fullw 2 : clearw 2
40    gosub modul1:gosub pause:clearw 2:gosub modul2:gosub pause
50    clearw 2:gosub modul3:gosub pause
60    clearw 2:gosub modul1:gosub modul2:gosub pause:end
70    rem-----------------------------------------------
80    '
90    modul1:
100   for x=120 to 520 step 4
110   linef x,200,640,x-120
120   linef 640-x,200,0,x-120
130   next
140   return
150   rem-----------------------------------------------
160   '
170   modul2:
180   for x=120 to 520 step 4
190   linef x,200,640,520-x
200   linef 640-x,200,0,520-x
210   next
220   return
230   rem-----------------------------------------------
240   '
250   modul3:
260   for x=120 to 520 step 4
270   linef x,200,0,520-x
280   linef 640-x,200,0,520-x
290   next
300   return
310   rem-----------------------------------------------
320   '
330   pause:
340   for i=1 to 8000 : next i : return
```

Häufig wird der Programmierer natürlich mehrere Spiegelachsen verwenden, und spätestens dann kommt ein weiterer Aspekt - zufällig aber doch sehr erwünscht - ins Spiel. Die Rede ist hier von Moire, ein Effekt, der seinen Namen nach einem Gewebe erhalten hat, welches durch seine spezielle Bearbeitung eine eigenartig schillernde Oberfäche aufweist.

Sie alle kennen diese Erscheinung auch von Ihrem Farbfernsehgerät her, wenn sich über eine feingemusterte Fläche einige Farbschlieren ziehen, die dort nichts zu suchen haben.

Dort findet sich die Bildröhre als der Verursacher für diesen Effekt, denn sie ist nicht mehr in der Lage, die feinen Details einwandfrei darzustellen. Dies gilt natürlich ebenso für einen an den ST angeschlossenem Farbmonitor, denn die in dessen Bildröhre befindliche Schlitzmaske setzt dem Auflösungsvermögen natürlich ebenso Grenzen:

```
10 rem Moire (Farbmonitor)
20 rem ------------------
30 clearw 2:fullw 2
40 for x=1 to 640 step 2
50 linef x,1,x,400
60 next
```

So kann es durch den physikalischen Aufbau der Röhre bedingt sogar zu Farbeffekten kommen; selbst völlig neue Farben können durch einen Mischeffekt entstehen, wenn der Elektronenstrahl einen Punkt anzusteuern versucht, der gerade von der Maske abgedeckt wird. Allerdings hängt dieser Effekt sehr stark vom benutzten Monitor und dessen Auflösungsvermögen ab und ist somit nicht kalkulierbar. Ist das Auflösungsvermögen niedrig - und wer wünscht sich das schon -, werden diese Effekte auftreten, ein hochwertiger Monitor wird jedoch in der Lage sein, zwei dicht nebeneinanderliegende Punkte auch als solche abzu-

bilden. Wie es um Ihren Farbmonitor steht (sofern Sie einen besitzen) zeigt Ihnen das folgende Programm, das ein Testbild, ähnlich dem der Fernsehstationen, erzeugt:

```
10    rem Demonstration: Aufloesung
20    rem ------------------------
30    clearw 2 : fullw 2
40    y1=100 : y2=300
50    gotoxy 1,19
60    print"                  20                    10       5     4
 3  2 1"
70    linef 80,100,550,100
80    linef 550,100,550,300
90    linef 550,300,80,300
100   linef 80,300,80,100
110   for x=80 to 290 step 21
120   gosub draw
130   next x
140   for x=290 to 400 step 11
150   gosub draw
160   next x
170   for x=400 to 450 step 5
180   gosub draw
190   next x
200   for x=450 to 490 step 4
210   gosub draw
220   next x
230   for x=490 to 520 step 3
240   gosub draw
250   next x
260   for x=520 to 540 step 2
270   gosub draw
280   next x
290   for x=540 to 550
300   gosub draw
310   next x
```

```
320   input dummy$
330   end
340   draw:
350   linef x,y1,x,y2 : return
```

Für den SM124 gilt zuvor gesagtes natürlich nicht uneingeschränkt. Wenn hier dennoch kompliziert gemusterte Grafiken entstehen obwohl nur einfache Linien ausgegeben wurden, dann liegt das vielmehr daran, daß in der Computertechnik eine Linie meistens gar keine Linie, sondern viel eher eine Treppe ist, wie Abbildung 10 zeigt.

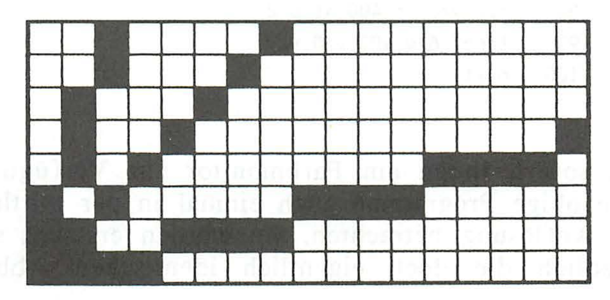


Abb. 10: Eine Gerade ist eine Treppe

Somit wird auch ein Strahlenbüschel kaum als solches erscheinen, sondern zu weitaus interessanteren Ergebnissen führen:

```
10    rem moire1
20    defint a-z
30    fullw 2 : clearw 2
40    for x=0 to 640 step 2
```

```
50    linef 0,0,x,400
60    linef 640,400,x,0
70    next
```

```
10    rem moire2
20    defint a-z
30    fullw 2 : clearw 2
40    for x=0 to 640 step 2
60    linef 640-x,400,x,0
70    next
80    for y=0 to 400 step 2
90    linef 640,400-y,0,y
100   next
```

Übrigens, sofern Ihnen ein Farbmonitor zur Verfügung steht, sollten Sie obige Programme auch einmal in der mittleren und niedrigen Auflösung betrachten. Sie werden erstaunt sein, wie unterschiedlich die doch eigentlich identischen Abbildungen wirken.

Überhaupt können kleine Änderungen andere Ergebnisse bedeuten, so sollten Sie probeweise auch einmal die Schrittweiten hinter Step abändern.

Um solche Grafiken auf einem beliebigen Computersystem erzeugen zu können, müssen Sie sich nur über dessen Koordinatensystem und über dessen Befehlsfolge zur Erzeugung einer Linie im klaren sein.

Mit diesem Wissen ausgestattet, können Sie dann jeweils durch kleine Änderungen zahllose voneinander verschiedene Grafiken erzeugen.

Denn die einzelnen Linien müssen ja nicht, wie bei obigen Beispielen, von den Eckpunkten des Bildschirmes ausgehen, sondern die Strahlen können an jedem beliebigen Punkt beginnen. Weiterhin kann es auch mehrere Ursprungspunkte geben, die

Strahlen können sich schneiden, oder die Schrittweite kann sich von Linie zu Linie ändern; unterschiedliche Farben innerhalb eines Bildes können ebenfalls zu dessen Gestaltung beitragen:

```
10    rem Grafik2
20    rem 1 Ursprungspunkt
30    rem ----------------
40    defint a-z
50    fullw 2
60    clearw 2
80    x=rnd(1)*640 : y=rnd(1)*400
90    schrittweite = rnd(1) * 10 + 2
100   for x1=1 to 640 step schrittweite
110   linef x,y,x1,400
120   linef x,y,x1,1
130   next x1
140   for y1=400 to 1 step - (schrittweite / 2)
150   linef x,y,640,y1
160   linef x,y,1,y1
170   next y1
190   for i=1 to 10000:next
200   goto 60
```

Wenn Sie nun Grafiken mit mehreren Ursprungspunkten erzeugen wollen, müssen Sie das Programm ein wenig abändern.

Ergänzen Sie eine weitere Schleife und legen Sie die Zahl der gewünschten Punkte mit dem Schleifenendwert in Zeile 70 fest. Möglicherweise empfiehlt es sich dann aber, die Mindestschrittweite zu erhöhen, so daß Ihr Kunstwerk nicht schwarz in schwarz verläuft.

```
70 FOR PUNKT=1 TO 2
180 NEXT PUNKT
200 CLEARW 2:GOTO 70
```

Um nun zum Thema - und auch wieder zu Logo - zurückzukommen: wie zuvor erwähnt, ist die Symmetrie nicht das einzige Gestaltungsmittel des Computergrafikers. Hinzu kommen noch die selbstidentischen Abbildungen. Darunter versteht man eine Grafik, die innerhalb eines Teilbereiches wieder das eigene Bild erhält. Später, wenn es um die Darstellung mathematischer Funktionen geht werden Sie dazu noch einige sehr schöne Beispiele kennenlernen. Hier reicht es aus, wenn Sie sich an die ersten Logo-Versuche erinnern. Dort bestand das gesamte Kunstwerk aus einer Reihe von einzelnen Quadraten, die mehrmals in verschiedener Position dargestellt wurden.

Programmtechnisch wird solch eine Vorgehensweise durch die Rekursion unterstützt: ein Programmteil ruft sich immer wieder selber auf. Und wenn dieser Programmteil dann einfache Grafiken erzeugt, kann das Gesamtprodukt eine eindrucksvolle Grafik sein.

Zwei Beispiele, die immer wieder angeführt werden, um die Rekursion begreiflich darzustellen, sollen hier zur Erläuterung ausreichen. Die einfachste Anwendung besteht in der rekursiven Darstellungsweise eines Kreises:

```
TO KREIS
RT 1
FD 1
KREIS
```

Nach 360 Aufrufen der Prozedur Kreis steht er auf dem Schirm und Sie können das Programm abbrechen. Allerdings, die wesentlichste Eigenschaft der Rekursion haben wir dabei noch gar nicht ausgenutzt. Denn bei jedem Aufruf der Subroutine legt der Computer die in der Prozedur befindlichen Variablen neu an, so daß Werte, die der gleichen Variablen beim zweiten Durchlauf der Prozedur zugewiesen werden, die alten Daten nicht überschreiben. Sie bleiben erhalten und stehen, nachdem der Rechner die Bearbeitung des zweiten Aufrufes völlig abgeschlossen hat, wieder zur Verfügung.

Die typischste Anwendung, die von rekursiven Datenstrukturen Gebrauch macht, nennt sich HILBERT, womit die Darstellung einer ganz bestimmten Funktion gemeint ist, um die zumindest kein Pascal-Anhänger herumkommt. Hier wurde die Hilbert-Kurve einmal in Logo implementiert, allerdings müssen Sie sich schon ein wenig gedulden, ehe der Programmlauf beendet ist:

```
TO A :ZAHL
IF (:ZAHL > 1)
   [D (:ZAHL - 1) SETH 270 FD :H
    A (:ZAHL - 1) SETH 180 FD :H
    A (:ZAHL - 1) SETH 90 FD :H
    B (:ZAHL - 1)]
END

TO B :ZAHL
IF (:ZAHL > 1)
   [C (:ZAHL - 1) SETH 0 FD :H
    B (:ZAHL - 1) SETH 90 FD :H
    B (:ZAHL - 1) SETH 180 FD :H
    A (:ZAHL - 1)]
END

TO C :ZAHL
IF (:ZAHL > 1)
   [B (:ZAHL - 1) SETH 90 FD :H
    C (:ZAHL - 1) SETH 0 FD :H
    C (:ZAHL - 1) SETH 270 FD :H
    D (:ZAHL - 1)]
END

TO D :ZAHL
IF (:ZAHL > 1)
   [A (:ZAHL - 1) SETH 180 FD :H
    D (:ZAHL - 1) SETH 270 FD :H
    D (:ZAHL - 1) SETH 0 FD :H
    C (:ZAHL - 1)]
END
```

```
TO START
NAME (:ORD + 1) "ORD
NAME (QUOTIENT :H 2) "H
NAME (:XX + (QUOTIENT :H 2)) "XX
NAME (:YY + (QUOTIENT :H 2)) "YY
PENUP
SETX :XX
SETY :YY
PENDOWN
A :ORD
END

TO HILBERT
CS
HT
CLEARTEXT
PR []
PR []
PR [Überlagerte Hilbert - Kurven]
PR []
PR [aus Algorithmen und Datenstrukturen]
PR [von Niklaus Wirth]
PR []
PR []
MAKE "H 256
MAKE "ORD 1
MAKE "XX 0
MAKE "YY 0
REPEAT 6 [START]
END

MAKE "ORD 8
MAKE "H 2
MAKE "YCOR 82
MAKE "XCOR 170
MAKE "YY 127
MAKE "XX 127
```

### 2.2.2.2 Logo-Grafikdemos

Damit sollten Ihnen die Grundzüge der Grafikprogrammierung - zumindest soweit es um Computerkunst geht - klar sein. Auf den folgenden Seiten finden Sie nun zahlreiche kurze Logo-Programme welche, überwiegend rekursiv, Bilder erzeugen, die Sie zum Teil sicherlich schon irgendwo gesehen haben. Ich hoffe, daß diese schnell einzugebenden Programme Ihnen als Anregung für eigene Experimente dienlich sind. Zwar werden einige Kenntnisse, die erst im nachstehenden Teil des Buches erläutert werden, bereits genutzt, z. B. die Berechnung einer Kreisbahn, dennoch wird Ihnen das Verständnis dieser Listings keine allzu großen Schwierigkeiten bereiten.

```
TO MOIRE
(LOCAL "X "Y "R "DW "W "Z)
HT CS FS
SETPAN [150 100]
SETZOOM 1.2
SETLINE [1 1 1]
MAKE "R 200
MAKE "DW 0.01
MAKE "W 0
MAKE "Z 180 / PI
MOIREZEICH
SETZOOM 1
SETPAN [0 0]
STOP
END

TO MOIREZEICH
IF NOT (:W < (2 * PI)) [STOP]
MAKE "X (:R * COS (:W * :Z))
MAKE "Y (:R * SIN (:W * :Z))
MAKE "W :W + :DW
MAKE "R :R * 0.997
MAKE "DW :DW / 0.997
```

```
LINEX 100 70 (100 + :X) (70 + :Y)
MOIREZEICH
END

TO LINEX :X1 :Y1 :X2 :Y2
PU
SETPOS LIST :X1 :Y1
PX
SETPOS LIST :X2 :Y2
END
```

```
TO WURFNETZ
(LOCAL "X "Y "XA "YA "VX "VY "V "W "I)
HT CS FS
SETPAN [150 -90]
SETZOOM 1.9
SETLINE [1 1 1]
MAKE "I 0
MAKE "V 17
WURFZEICH1
SETZOOM 1
SETPAN [0 0]
STOP
END

TO WURFZEICH1
IF :I > 30 [STOP]
MAKE "W :I * 3
MAKE "VX :V * COS (:W)
MAKE "VY :V * SIN (:W)
MAKE "X 0
MAKE "Y 0
MAKE "XA 0
MAKE "YA 0
MAKE "VY :VY - 0.5
WURFZEICH2
MAKE "I :I + 1
WURFZEICH1
END

TO WURFZEICH2
IF :Y =< -30 [STOP]
POLY (LIST :XA -(170 - :YA) :X -(170 - :Y))
MAKE "XA :X
MAKE "YA :Y
MAKE "X :X + :VX
MAKE "Y :Y + :VY
MAKE "VY :VY - 1
WURFZEICH2
END
```

```
TO TRICHTER
HT CS
SETZOOM 1.5
SETPAN [150 100]
SETLINE [1 1 1]
TZEICH1 0 24 160
TZEICH2 24 48 208
TZEICH1 48 72 112
TZEICH2 72 96 256
SETZOOM 1
SETPAN [0 0]
STOP
END

TO TZEICH1 :R :E :P
IF :R > :E [STOP]
CIRCLE (LIST (:P + :R) 100 :R)
TZEICH1 (:R + 2) :E :P
END

TO TZEICH2 :R :E :P
IF :R > :E [STOP]
CIRCLE (LIST (:P - :R) 100 :R)
TZEICH2 (:R + 2) :E :P
END
```

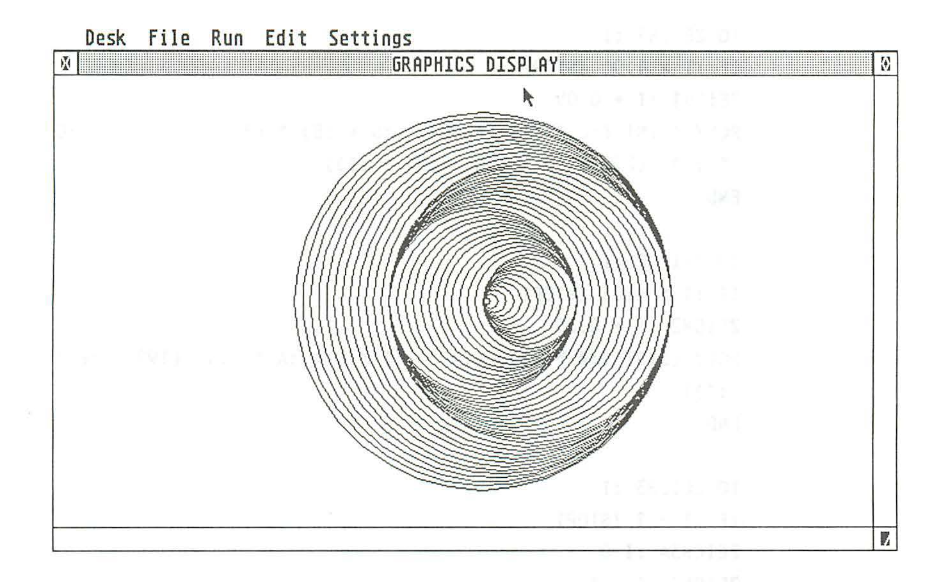



Abb. 11: Hardcopy zu TRICHTER

```
TO PARABOLOID
(LOCAL "A "B "C "D)
HT CS
SETZOOM 1.5
SETPAN [150 -100]
SETLINE [1 1 1]
MAKE "A 250
MAKE "B 150
MAKE "C 70
MAKE "D 50
ZEICH1 0
ZEICH2 0
BOX [0 -150 250 150]
BOX [70 -200 250 150]
ZEICH3 0
SETZOOM 1
SETPAN [0 0]
STOP
END
```

```
TO ZEICH1 :I
IF :I > 1.01 [STOP]
ZEICH1 :I + 0.05
POLY (LIST (:C * :I) -(200 - (:D + :B) * (1 - :I)) (:A + :C
 * :I) -(200 - :D - (:B - :D) * :I))
END

TO ZEICH2 :I
IF :I > 1.01 [STOP]
ZEICH2 :I + 0.05
POLY (LIST (:A * :I) -(:B * :I) (:C + :A * :I) -(199 - :B *
 :I))
END

TO ZEICH3 :I
IF :I > 1 [STOP]
ZEICH3A :I 0
ZEICH3 :I + 1
END

TO ZEICH3A :I :J
IF :J > 1 [STOP]
POLY (LIST (:I * :A) -(:J * :B) (:I * :A + :C) -(:J * :B +
 :D))
ZEICH3A :I :J + 1
END
```

```
TO EURO
IF :W1 > (2 * PI) [STOP]
MAKE "W2 (4 * :W1)
MAKE "XA (1.8 * ((:R1 * SIN (:W1 * 180 / PI))
MAKE "XB (:R2 * SIN (:W2 * 180 / PI))
MAKE "X1 :XA + :XB
MAKE "X2 :XA - :XB
MAKE "YA (:R1 * COS (:W1 * 180 / PI))
MAKE "YB (:R2 * COS (:W2 * 180 / PI))
MAKE "Y1 (:YA + :YB) * 2
MAKE "Y2 (:YA - :YB) * 2
POLY (LIST :X1 :Y1 :X2 :Y2)
MAKE "W1 :W1 + 0.05
EURO
END

TO EUROVISION
(LOCAL "X1 "X2 "XA "XB "Y1 "Y2 "YA "YB "R1 "R2 "W1 "W2)
HT FS CS
SETLINE [1 1 1]
MAKE "R1 70
MAKE "R2 30
MAKE "W1 0
EURO
STOP
END
```

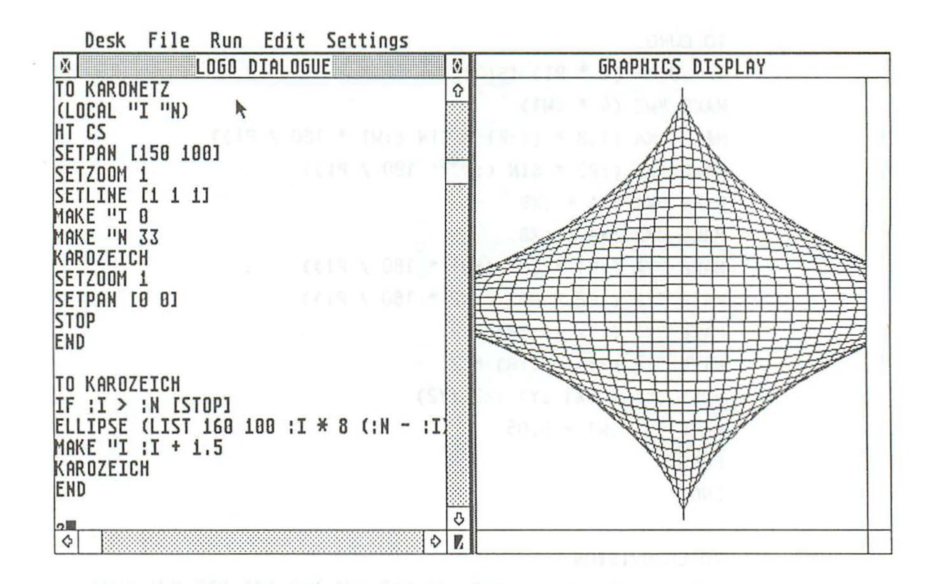



Abb. 12: Das Programm KARONETZ

```
TO DREYSTART
CS
SETLINE [1 1 1]
SETCURSOR -160 -95
SETHEADING 96
DREYFUSS 330 122 3
WAIT 100
CS
SETCURSOR -160 -75
SETHEADING 99
DREYFUSS 320 123 3
WAIT 100
CS
SETCURSOR -160 -75
SETHEADING 99
DREYFUSS 320 123 2
WAIT 100
STOP
END
```

```
TO DREYFUSS :LAE :DEG :SUB
IF :LAE < 0 [STOP]
FD :LAE LT :DEG
DREYFUSS (:LAE - :SUB) :DEG :SUB
END

TO SETCURSOR :X :Y
 PU HOME
 SETPOS LIST :X :Y
 PD
 STOP
END

TO WAIT :ZEIT
IF :ZEIT < 1 [STOP]
WAIT :ZEIT - 1
END
```

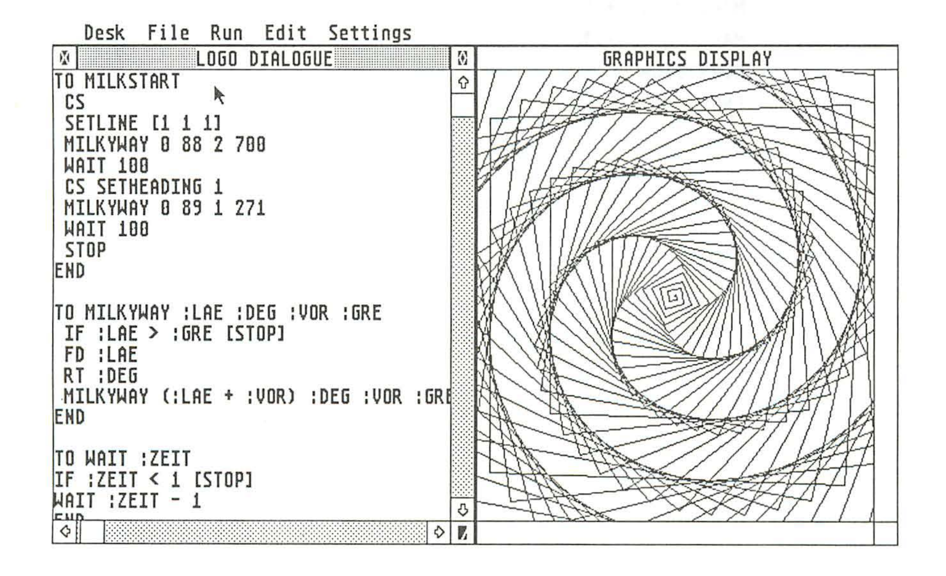



Abb. 13: Das Programm MILKYWAY

```
TO FUELLHORN
(LOCAL "X "Y "R "W)
HT CS FS
SETPAN [150 100]
SETZOOM 1
SETLINE [1 1 1]
MAKE "R 80
MAKE "W 0
HORNZEICH
SETZOOM 1
SETPAN [0 0]
STOP
END

TO HORNZEICH
IF :R =< 0 [STOP]
MAKE "X 160 + 110 * COS (:W * 180 / PI)
MAKE "Y 110 - 110 * SIN (:W * 180 / PI)
CIRCLE (LIST :X :Y :R)
MAKE "W :W + 0.1
MAKE "R :R - 1
HORNZEICH
END
```

```
TO GUILLOCH
(LOCAL "A "B "X "Y "D "XO "YA "XA "W)
HT CS FS
SETPAN [150 100]
SETZOOM 1.5
SETLINE [1 1 1]
MAKE "A 100
MAKE "B 20
MAKE "X 25
MAKE "D 0.05
MAKE "XO 20
MAKE "YA 100
MAKE "XA :XO
MAKE "W 0
GUILZEICH
SETZOOM 1
SETPAN [0 0]
STOP
END

TO GUILZEICH
IF :XO >= 310 [STOP]
MAKE "XO :XO + :D
MAKE "X :XO + :B * SIN (2 * :W * 180 / PI)
MAKE "Y 100 + :A * SIN (:W * 180 / PI)
POLY (LIST :XA :YA :X :Y)
MAKE "A :A * 0.9998
MAKE "B :B * 0.9998
MAKE "XA :X
MAKE "YA :Y
MAKE "W :W + 0.05
GUILZEICH
END
```

```
TO SNOWSTART
CS
SETCURSOR -30 150
SETTEXT 29
TT [SNOWSTAR]
SETTEXT 0
SETLINE [1 1 1]
SETCURSOR -70 -130
(LOCAL "SIZE "LEVEL)
MAKE "SIZE 250 MAKE "LEVEL 4
SNOW :SIZE :LEVEL
WAIT 100
STOP
END

TO SNOW :SIZE :LEVEL
IF :LEVEL < 1 [STOP]
S :SIZE :LEVEL
SNOW :SIZE :LEVEL - 1
END

TO SIDE :SIZE :LEVEL
IF :LEVEL = 0 [FORWARD :SIZE STOP]
SIDE (:SIZE / 3) (:LEVEL - 1)
LEFT 60
SIDE (:SIZE / 3) (:LEVEL - 1)
RIGHT 120
SIDE (:SIZE / 3) (:LEVEL - 1)
LEFT 60
SIDE (:SIZE / 3) (:LEVEL - 1)
END

TO S :SIZE :LEVEL
REPEAT 3 [SIDE :SIZE :LEVEL RIGHT 120]
STOP
END
```

```
TO SETCURSOR :X :Y
 PU HOME
 SETPOS LIST :X :Y
 PD
 STOP
END

TO WAIT :ZEIT
IF :ZEIT < 1 [STOP]
WAIT :ZEIT - 1
END
```

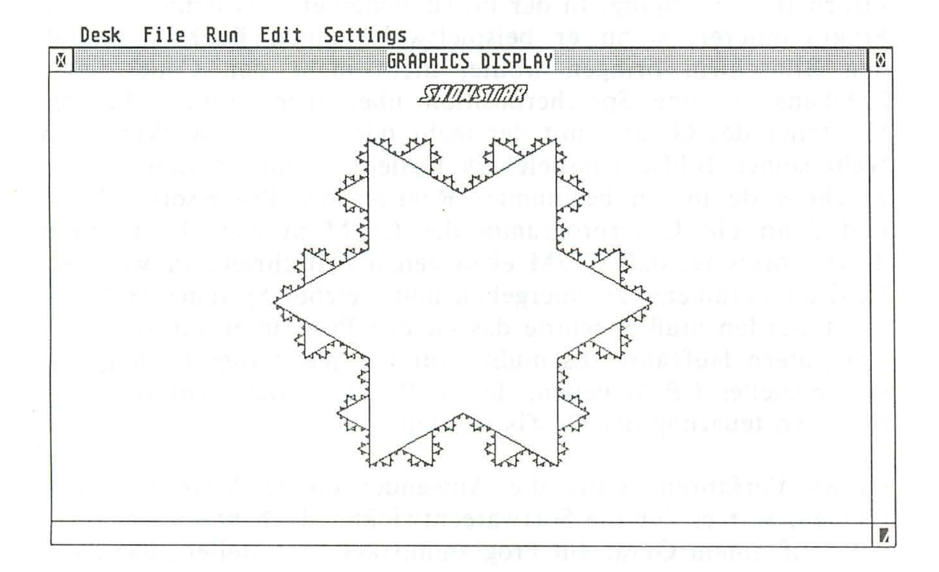



Abb. 14: Hardcopy zu SNOWSTAR

## 2.3 Grafik unter GEM

Wer sich mit den Logo-Programmen ein wenig beschäftigt hat, der ist auf einige spezielle Grafikanweisungen gestoßen, die den ST aus der Menge der anderen Computer hervorheben. Handelt es sich dabei doch weniger um Routinen, die die Logo-Entwickler implementiert haben, als vielmehr um Unterprogramme des Betriebssystemes, zu dem wir neben GEMDOS auch GEM zählen. Speziell ist es die Bibliothek des Virtual Device Interfaces, eine Programmsammlung, die eine virtuelle Geräteschnittstelle darstellt und eigentlich mit allem zu tun hat, was auf dem Bildschirm oder sonstwo ausgegeben werden soll.

Die grundlegende Idee, die zur Entwicklung von GEM und somit auch des VDI geführt hat, ist die gleiche, der wir auch schon das Betriebssystem CP/M verdanken. Denn CP/M machte den Programmierer im wesentlichen von der vorhandenen Hardware unabhängig - soweit es die Verarbeitung von textuellen Informationen anging. In der Praxis bedeutete das dann, daß der Programmierer, wenn er beispielsweise einen Buchstaben auf den Bildschirm bringen wollte, nicht mehr den Code dieses Zeichens an eine Speicherposition übergeben mußte, die der Hersteller des Gerätes mit der mehr oder weniger willkürlichen Wahl seines Bildschirmspeichers festlegte, sondern daß er den Zeichencode in ein bestimmtes Register des Prozessors schrieb und dann ein Unterprogramm des CP/M aufrief. Und durch diese Normung, daß CP/M eben genau festschrieb, in welchem Register Parameter zu übergeben und welcher Systemaufruf benutzt werden mußte, wurde das gleiche Programm auf zahllosen Computern lauffähig. Es mußte nur für jeden dieser Computer ein spezielles CP/M geben, dessen Routinen dann für die korrekte Ansteuerung der Grafik sorgten.

Dieses Verfahren stellte die Anwender einige Jahre lang zufrieden, war es für die Softwareentwickler doch besonders reizvoll, auf einem Gerät ein Programmpaket zu erstellen, das dann auf zahllosen Computern lauffähig war.

Doch ist in den letzten Jahren, bedingt durch die gesteigerten technischen Möglichkeiten, ein großes Interesse an grafischer

Datenverarbeitung erwacht. Die Computer wurden grafikfähig, doch da jeder Hersteller sein eigenes Süppchen kochte, war ein Softwareentwickler gezwungen, sich für ein bestimmtes Gerät zu entscheiden und speziell für dieses eine ein Programm zu schreiben. Eine Anpassung auf ein anderes Gerät war immer mit erheblichem Aufwand verbunden. Und auch der einzelne Anwender war mit den Ergebnissen seiner EDV nicht immer ganz zufrieden: Hatte er sich beispielsweise einen grafikfähigen Drucker angeschafft, so zeigte sich -bedingt durch unterschiedliche Auflösungen in x- und y-Richtung zwischen Computer und Drucker - ein Kreis auf dem Abbild des Bildschirminhaltes meistens als Ellipse und ein Quadrat als Rechteck. Was unbedingt benötigt wurde, war eine softwaremäßige Schnittstelle, die in der Lage sein sollte, verschiedene Geräte anzusteuern und für die Korrektheit des Abbildungsverhältnisses zu sorgen. Darüber hinaus sollte sie auch noch einige wichtige Zeichenroutinen (Set Point, Draw Line) enthalten, die ähnlich der CP/M-Aufrufe gehandhabt werden konnten.

Aus diesen Überlegungen heraus entstand zunächst GSX, die Graphics System Extension. Es handelte sich dabei um eine Sammlung von Routinen zum gemeinsamen Gebrauch mit CP/M. Die konsequente Weiterentwicklung dieser Programmsammlung zum kompletten Grafik-Betriebssystem bescherte uns schließlich GEM - den Graphics Environment Manager.

Mit GEM steht endlich eine grafikfähige Ein- und Ausgabeschnittstelle bereit, die zudem auch noch so komfortabel ist, daß sie dem Bedürfnis der Anwender nach benutzerfreundlichen Schnittstellen entspricht. GEM enthält zahlreiche Unterprogramme zum Zeichnen von Linien, Kreisen, und Polygonzügen, zur Ausgabe von Text und auch Routinen zur Ansteuerung der unterschiedlichsten Peripheriegeräte.

Logisch, daß ein Programmierer diese Unterprogramme auch nutzt und nicht statt dessen das Rad zum zweiten Mal erfindet. Und so ist es auch nicht weiter verwunderlich, wenn sich die grafischen Fähigkeiten von BASIC und Logo ähneln. Denn hier wie dort finden wir Anweisungen zur Einstellung von Linien-

typen und Schreibmodi. Ebenso wie im BASIC vermissen wir auch in Logo einen Befehl, der das Setzen eines einzelnen Punktes erlaubt!

### 2.3.1 GEM

Betrachten wir nun kurz die Arbeitsweise von GEM, insbesonders natürlich des VDI. Welchen zusätzlichen Aufwand erfordert eine GEM-Applikation, und, vor allem, wie können wir die GEM-Routinen nutzen?

Nun, zunächst einmal wird das VDI bei Arbeitsbeginn erst initialisiert werden müssen. Schließlich muß es auf das speziell anzusteuernde Gerät eingestellt werden. Handelt es sich um ein Rastergerät (Bildschirm) oder erwartet es zu seiner Ansteuerung Vektoren (Plotter)? Wie viel Punkte kann es waagrecht und senkrecht auf einer Strecke von einem cm abbilden? Wie vieler Zeichenstifte (= Farben) kann sich das anzusteuernde Gerät bedienen?

Diese und viele andere Informationen über das zu benutzende Gerät enthält jeweils ein spezieller Gerätetreiber. Hierbei handelt es sich um den hardwareabhängigen Teil des GEM, der bei Bedarf geladen wird. Standard für den ST ist hier der Treiber für den Bildschirm, der übrigens bereits ins TOS integriert wurde. Somit kann der Bildschirm unter Ausnutzung sämtlicher Fähigkeiten des GEM angesteuert werden. Um einen Drucker ebenso ansprechen zu können, müßten Sie über einen Gerätetreiber für genau diesen Drucker verfügen und diesen im ersten Arbeitsschritt laden.

Um nun den zweiten wichtigen Gesichtspunkt zu berücksichtigen und die Abbildung bei Beibehaltung der Proportionen gewährleisten zu können, ermöglicht GEM das Arbeiten mit zwei unterschiedlichen Koordinatensystemen, nämlich mit dem

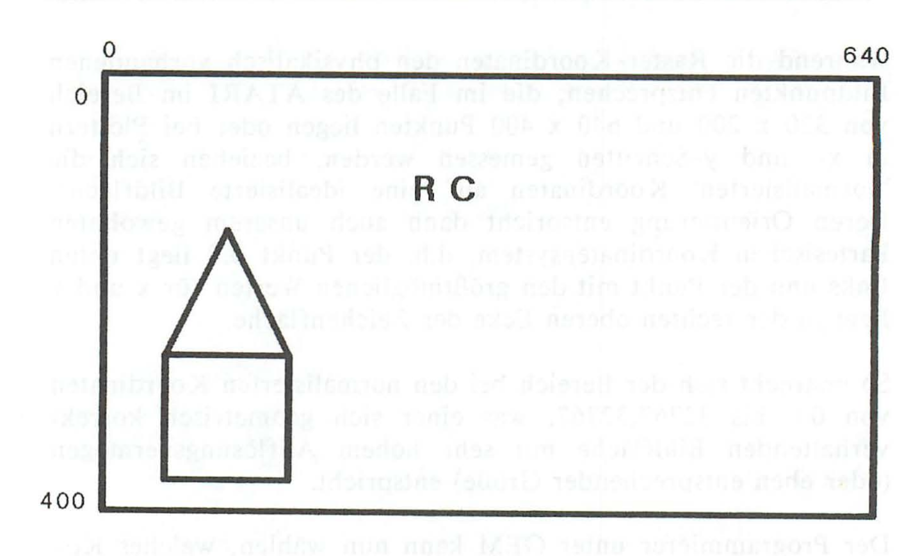
- NDC, Normalized Device Coordinates, und
- RC, den Raster Coordinates.

Während die Raster-Koordinaten den physikalisch vorhandenen Bildpunkten entsprechen, die im Falle des ATARI im Bereich von 320 x 200 und 640 x 400 Punkten liegen oder bei Plottern in x- und y-Schritten gemessen werden, beziehen sich die 'normalisierten' Koordinaten auf eine idealisierte Bildfläche. Deren Orientierung entspricht dann auch unserem gewohnten kartesischen Koordinatensystem, d.h. der Punkt 0,0 liegt unten links und der Punkt mit den größtmöglichen Werten für x und y liegt in der rechten oberen Ecke der Zeichenfläche.

So erstreckt sich der Bereich bei den normalisierten Koordinaten von 0,0 bis 32767,32767, was einer sich geometrisch korrekt verhaltenden Bildfläche mit sehr hohem Auflösungsvermögen (oder eben entsprechender Größe) entspricht.

Der Programmierer unter GEM kann nun wählen, welches Koordinatensystem er benutzen möchte. Entscheidet er sich für NDC, so werden die eingegebenen Werte während der Programmausführung vom VDI auf die wirklich vorhandenen Koordinaten umgerechnet. Wird ein Quadrat der Länge 100 eingegeben, so erscheint es auch als Quadrat auf dem Monitor. Hat er sich hingegen für die Rasterkoordinaten entschieden, so nimmt das VDI keinerlei Umrechnungen vor. Der Programmierer beziehungsweise das laufende Programm muß selbst sämtliche Maßstabsveränderungen beachten.

Vorteile bringt das NDC-System in Verbindung mit dem VDI vor allem beim Austausch von Grafiken zwischen unterschiedlichen Peripheriegeräten. So werden wir, wie zuvor schon erwähnt, mit den üblichen Rasterkoordinaten und der bislang gewohnten Ansteuerung von Druckern immer unsere Schwierigkeiten haben,



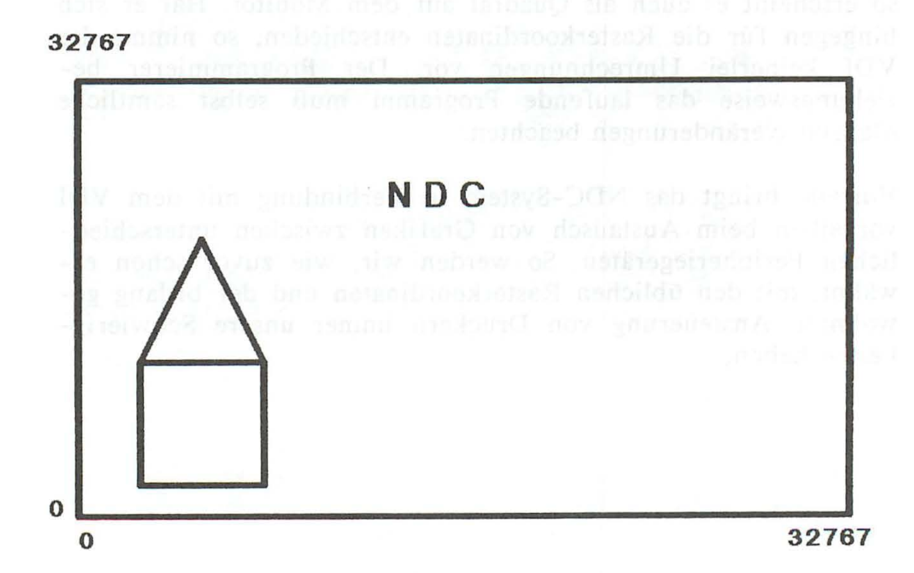




Abb. 15: Der Unterschied zwischen RC und NDC

besonders was das Verhältnis zwischen Länge und Breite der Abbildung betrifft. Denn bei einem Koordinatensystem von 600 x 200 wird das Verhältnis zwischen den Koordinaten auf dem Bildschirm ungefähr 1:1.8 betragen, hier als Quadrat erscheinende Rechtecke werden auf einem Drucker, sofern er nicht zufällig mit genau dem gleichen System arbeitet, niemals mehr als Quadrat erscheinen.

Doch im Falle einer Adressierung an das VDI kann der Geräte-Treiber die notwendigen Umrechnungen vornehmen; Darstellungen gleich welcher Art werden demnach auf sämtlichen Peripheriegeräten gleich in Erscheinung treten. Als nachteilig kann sich dabei jedoch der erhöhte Zeitbedarf solcher Programme erweisen, schließlich muß die Lage eines jeden einzelnen Grafikpunktes erst auf das aktuelle Koordinatensystem umgerechnet werden. Aus diesem Grunde empfiehlt es sich eigentlich, immer mit den Rasterkoordinaten zu arbeiten, es sei denn, Sie müssen unbedingt portable Programme schreiben. Dies wird jedoch nur bei größeren Softwarehäusern der Fall sein, und so werden auch wir bei unseren späteren Programmen stets die Rasterkoordinaten benutzen. Im übrigen trifft diese Unterscheidung sowieso nur für den Gebrauch von Hochsprachen wie C, Pascal und Modula zu, denn für den BASIC- oder Logoprogrammierer hat der Interpreter die Initialisierung der Arbeitsstation, und somit auch die Einstellung des Koordinatensystems, vorweggenommen.

### 2.3.2 Die Nutzung von GEM-Routinen

GEM wird die Grafikprogrammierung also wesentlich vereinfachen; es stellt sich dem Benutzer als Black-Box dar, d.h., dieser muß nur noch wissen, was GEM kann und wie er GEM dazu bringt, das Gewünschte zu tun.

Nun, bei der immensen Leistungsfähigkeit von GEM kann man sich leicht vorstellen, daß einige wenige Prozessorregister nicht ausreichend sind, um die benötigte Anzahl von Parameteren zu übergeben. Daher hat man einen Speicherbereich definiert, der als Array gehandhabt wird und dessen einzelne Elemente für

den Datenaustausch zwischen Anwendung und GEM zuständig sind.

Genau genommen handelt es sich sogar um fünf Arrays, die vor dem Aufruf einer VDI-Routine initialisiert werden müssen:

- contrl
- intin
- intout
- ptsin
- ptsout

Es handelt sich dabei um Integer-Arrays, Dezimalzahlen wären ja auch recht sinnlos. Die Abkürzungen, die Sie sich unbedingt merken müssen, weil sie in jeder Sprache (C, Pascal usw., aber auch BASIC) bekannt sind, stammen übrigens von den Bezeichnungen Control-Array, Integer Input Array, Integer Output Array, Point Coordinate Input Array und Point Coordinate Output Array. Diese Namen lassen durchaus auf die Funktion der zugehörigen Arrays schließen.

So enthält das Kontroll-Array alle Daten, die die gewünschte Aktion spezifizieren. In Contrl(0) muß immer der Operationscode der Funktion vor dem GEM-Call abgelegt werden (dieser OP-Code ist vergleichbar mit dem Ihnen vielleicht bekannten Funktionscode eines BDOS-Aufrufes bei CP/M).

Beispielsweise hat die Funktion zur Ausgabe eines Strings in einer spezifizierten Schriftart den Code 8, der Subroutine, die einen Polygonzug zeichnet, ist der Op-Code 6 zugewiesen usw. Die restlichen Variablen des Kontroll-Arrays enthalten dann Informationen über die Anzahl der Daten innerhalb der anderen Arrays oder aber auch Angaben zum adressierten Gerät. Beispielsweise ermittelt GEM für jedes geöffnete Fenster eine

Nummer, das sogenannte (Window-)Handle, die ebenfalls angegeben werden muß, um die Funktion am richtigen Gerät auszulösen.

Im Input-Array werden Parameter übergeben, mit denen der Aufruf ausgeführt werden soll. Im angeführten Beispiel der Stringausgabe wäre das die auszugebende Zeichenkette.

Intout enthält nach dem Aufruf die ermittelten Funktionswerte - natürlich nur dann, wenn die Bereitstellung irgendwelcher Informationen über die Funktion auch sinnvoll ist. So erhält der Aufrufer der Funktion Open Workstation, die wie im vorangegangenen Text erwähnt VDI auf die Arbeit mit einem bestimmten Gerät einstellt, innerhalb des Feldes von intout(0) bis intout(44), fünfundvierzig verschiedene Informationen über das Gerät!

Ptsin und ptsout schließlich sind Koordinatenangaben vorbehalten. Dabei kann es sich um eine Reihe von x-/y- Koordinaten (beispielsweise Parameter zum Zeichnen eines Polygonzuges), die dann meistens als pxyarray() bezeichnet werden, handeln oder aber auch um eine einzelne Koordinatenangabe wie im Falle der Funktion 8, die dann angibt, an welcher Stelle der Text erscheinen soll.

Anhand von zwei Beispielen, erstens an einem Open_workstation-Call und zweitens an v_gtext, soll die Arbeitsweise der GEM-Bibliothek im folgenden verdeutlicht werden.

#### 2.3.2.1 Beispiel 1: Initialisierung der Arbeitsstation

Open_workstation, op-code 1, ist der Aufruf, der zu Beginn eines jeden Sourcecodes für Compilersprachen zu stehen hat.

Im GEM-Handbuch findet man den Aufruf definiert als:

contrl(0) opcode = 1
contrl(1) Anzahl der Eingabedaten in ptsin() = 0
contrl(3) Anzahl der Eingabedaten in intin() = 11

intin(0) Gerätenummer; legt Treiber fest
intin(1) Linientyp
intin(2) Farbe bei Linienzügen
intin(3) Markertyp
intin(4) Polymarkerfarbe
intin(5) Schriftart
intin(6) Schriftfarbe
intin(7) Füllflächenbegrenzung
intin(8) Füllmuster
intin(9) Füllfarbe
intin(10) Koordinatensystem

Die Arraywerte müssen nun nach Wunsch des Anwenders initialisiert werden. So bedeutet eine 1 in intin(1), daß Linien durchgezogen und nicht strichliert werden; eine 0 in intin(10) würde NDC wählen, wogegen eine 2 hier RC einstellen würde (1 ist für zukünftige Anwendungen reserviert).

Es ist für eine jede Anwendung unabdingbar, hier die Betriebsart einzustellen, glücklicherweise jedoch haben die Entwickler von GEM die Parameter so vergeben, daß eine Eins immer der Normaleinstellung entspricht.

Aus diesem Grunde kann die gesamte Initialisierung auch innerhalb einer einzigen Schleife geschehen. Wie solch eine Schleife beispielsweise in C oder Modula-2 auszusehen hat, können Sie den hier folgenden Listings entnehmen.

```
open_vwork()
{
int i;
        for (i = 1; i <10; i++){
          int_in[i] = 1;
        }
        int_in[10] = 2;
        v_opnvwk(int_in, &handle, int_out);
}
```

```
FROM VDIControls IMPORT OpenVirtualWorkstation,

VAR In:VDIWorkInType;
    Out:VDIWorkOutType;

PROCEDURE open_vwork;
VAR i:INTEGER;
BEGIN
  FOR i:=0 TO 9 DO In[i]:=1 END;
  In[10]:=2;
  OpenVirtualWorkstation(In,handle,Out);
END;
```

### 2.3.2.2 Beispiel 2: Grafik-Textausgabe:

Das zweite Beispiel soll die Textausgabe an einer Grafikposition (nicht am Cursor!) demonstrieren. Mit der Funktion 8: v_gtext(handle,x,y,string) erlaubt GEM beispielsweise das korrekte Beschriften von Achsen bei Funktionsplotprogrammen.

Handle in der oben angegebenen C-Syntax weist dabei auf das Arbeitsgerät, string ist der auszugebende String, und x und y legen die Anfangskoordinaten für die Ausgabe fest.

Da es sich dabei um Pixelkoordinaten handelt, sollten diese mit der Höhe (bzw. Breite der Zeichen) multipliziert werden um

Zeilen und Spalten adressieren zu können. Die aktuelle Pixelhöhe des aktivierten Charaktersatzes wird als zweiter Parameter bei graf_handle übergeben.

Beispielaufruf:

```
handle=graf_handle(&dummy,&height,&dummy,&dummy);
v_gtext(handle,0,height*5,"Ist ja super");
```

Die Schriftart wird gewählt mit vst_effects(handle, effect). Der Wert von effect wird in intin(0) abgelegt und legt dabei die Schriftart für die nächsten Ausgaben fest. Deren Wahl wird getroffen durch Setzen der Bits 0 bis 5:

| BIT | zuständig für | GESETZT (=1) | GELÖSCHT (=0) |
|---|---|---|---|
| 0 | Fettdruck | fett | normal |
| 1 | Intensität | abgeschwächt | normal |
| 2 | Schrägschrift | Italics | normal |
| 3 | Unterstreichen | unterstrichen | normal |
| 4 | Umrissen | umrissen | normal |
| 5 | Schatten | mit Schatten | normal |

Die durch die entsprechend gesetzten Bits repräsentierte Zahl wird für effect eingesetzt. So bedeutet eine

| | | | |
|---|---|---|---|
| 1 | Fettdruck | denn 1 = | 000001 |
| 2 | halbe Helligkeit | denn 2 = | 000010 |
| 3 | fett und halbhell | | 000011 |
| 4 | schräg | | 000100 |
| 5 | schräg, fett | | 000101 |
| 6 | schräg, halbhell | | 000110 |
| 7 | schräg, halbhell, fett | | 000111 |
| 8 | unterstrichen, fett | | 001001 |
| 9 | unterstrichen, halbhell | | 001010 |

usw.

Welche Schriftarten GEM, und somit auch unser Atari, kennt, zeigt das folgende kurze Logoprogramm (denn Settype ist nichts anderes als ein Aufruf von vst_effects() von Logo aus):

```
TO TRUF :LINE
 IF :LINE > 3 [STOP]
 MAKE "XC (-285 + :LINE * 150)
 SETCURSOR :XC "74
 MAKE "HI :HI + 1
 MAKE "HOE :HI
 TZEICH :HOE * 16
 TRUF :LINE + 1
END

TO TZEICH :HOE
 IF (:ZAE > (:HOE - 1)) [STOP]
 SETTEXT :ZAE
 TT (LIST "Schriftbild :ZAE)
 RETURN
 MAKE "ZAE :ZAE + 1
 TZEICH :HOE
END
```

```
TO SETCURSOR :X :Y
 PU HOME
 SETPOS LIST :X :Y
 PD
 STOP
END

TO RETURN
 SETCURSOR XCOR YCOR - 16
 STOP
END

TO WAIT :ZEIT
IF :ZEIT < 1 [STOP]
WAIT :ZEIT - 1
END

TO SCHRIFTDEMO
(LOCAL "HI "HOE "XC "ZAE)
CS HT FS
SETTEXT 17
SETCURSOR -100 140
TT [SCHRIFTARTEN UNTER LOGO]
MAKE "ZAE 0
MAKE "HI 0
TRUF 0
WAIT 500
SETTEXT 0
END
```

Desk File Run Edit Settings

GRAPHICS DISPLAY

SCHRIFTARTEN UNTER LOGO

| | | | |
|---|---|---|---|
| Schriftbild 0 | Schriftbild 16 | Schriftbild 32 | Schriftbild 48 |
| Schriftbild 1 | Schriftbild 17 | Schriftbild 33 | Schriftbild 49 |
| Schriftbild 2 | Schriftbild 18 | Schriftbild 34 | Schriftbild 50 |
| Schriftbild 3 | Schriftbild 19 | Schriftbild 35 | Schriftbild 51 |
| Schriftbild 4 | Schriftbild 20 | Schriftbild 36 | Schriftbild 52 |
| Schriftbild 5 | Schriftbild 21 | Schriftbild 37 | Schriftbild 53 |
| Schriftbild 6 | Schriftbild 22 | Schriftbild 38 | Schriftbild 54 |
| Schriftbild 7 | Schriftbild 23 | Schriftbild 39 | Schriftbild 55 |
| Schriftbild 8 | Schriftbild 24 | Schriftbild 40 | Schriftbild 56 |
| Schriftbild 9 | Schriftbild 25 | Schriftbild 41 | Schriftbild 57 |
| Schriftbild 10 | Schriftbild 26 | Schriftbild 42 | Schriftbild 58 |
| Schriftbild 11 | Schriftbild 27 | Schriftbild 43 | Schriftbild 59 |
| Schriftbild 12 | Schriftbild 28 | Schriftbild 44 | Schriftbild 60 |
| Schriftbild 13 | Schriftbild 29 | Schriftbild 45 | Schriftbild 61 |
| Schriftbild 14 | Schriftbild 30 | Schriftbild 46 | Schriftbild 62 |
| Schriftbild 15 | Schriftbild 31 | Schriftbild 47 | Schriftbild 63 |

Abb. 16: Dieser Schriftarten bedient sich GEM

### 2.3.2.3 GEM-Calls von BASIC aus

Ich nehme an, es wird Sie nun interessieren, wie Sie diese und andere Aufrufe von BASIC aus tätigen können. Eigentlich müßte Ihr Wissen dazu bereits ausreichen. Denn alles, was Sie tun müssen, ist, die Arrays korrekt zu initialisieren und dann den VDI-Call auszuführen. Sofern Sie zu den Aufsteigern gehören, so werden Sie sicherlich wissen, daß der Aufruf einer Unterroutine in Maschinensprache immer über deren Adresse erfolgt. Doch welche Startadresse hat ein GEM-Call?

Glücklicherweise ist dieses Wissen im Falle des ST jedoch gar nicht erforderlich. Denn das ST-BASIC kennt nicht nur die Arrays, sondern es kennt auch einen Aufruf vdisys - und dieser übergibt die Kontrolle an die in contrl(0) spezifizierte VDI-Routine.

So lauten die Aufruf-Parameter für vst_effects:

| | |
|---|---|
| contrl(0) | Opcode = 106 |
| contrl(1) | Anzahl Werte in ptsin = 0 |
| contrl(3) | Anzahl Werte in intin = 1 |
| contrl(6) | Gerätenummer |
| | |
| intin(0) | Nummer der Schriftart |

Als BASIC-Programmierer haben Sie nun nichts anderes zu tun, als diese Werte an ihre Plätze zu poken.

Wohlgemerkt, poken; denn die Arrays sind in BASIC nicht als Arrays realisiert, sondern als zusammenhängender Speicherbereich, dessen Startadresse jeweils in contrl usw. festgelegt ist.

Ein GEM-Call von BASIC wird also immer ähnlich nachstehendem Beispiel aussehen:

```
10    rem Aufruf von Gemfunktion 106: Text Special Effects
20    '
100   poke contrl,106    : rem Funktionsnummer
110   poke contrl,106    : rem Funktionsnummer
120   poke contrl+2,0    : rem Anzahl Werte in ptsin
130   poke contrl+4,1    : rem Anzahl Werte in intin
140   poke contrl+6,1    : rem handle-Nummer: BASIC = 1
150   poke intin,3       : rem Schriftart: 3 = halbe
 Intensität
160   vdisys
170   print "Testing Schriftart"
```

Mit diesem Wissen ausgestattet fällt es uns nun nicht mehr schwer, ein ähnliches Programm auch in BASIC zu formulieren:

```
10    rem schrift.bas
20    rem Demonstration aller Schriftarten
30    rem --------------------------------
40    clearw 2 : fullw 2
50    for schriftart=0 to 31
60    gosub schriftbild
70    print "Schriftbild ";schriftart;" "
80    next schriftart
90    schriftart = 0 : gosub schriftbild : end
100   schriftbild:
110   poke contrl,106   : rem Funktionsnummer
120   poke contrl+2,0   : rem Anzahl Werte in ptsin
130   poke contrl+4,1   : rem Anzahl Werte in intin
140   poke contrl+6,1   : rem handle-Nummer: BASIC = 1
150   poke intin,schriftart
160   vdisys
170   return
```

# 3. Kapitel

# 2-D-Grafik

## 3.1 Von Punkten, Linien & Kreisen

Trotz allem bleibt auch bei Gebrauch der GEM-Routinen immer noch ein Manko. So haben wir bereits festgestellt, daß BASIC keinen Befehl zum Setzen eines Punktes kennt! Und zwar ganz einfach deshalb, weil diese Routine nicht in GEM enthalten ist. Dafür verfügen allerdings sämtliche Sprachen über die Möglichkeit zur Darstellung von Polygonzügen, eben weil das VDI nach Eingabe einer Anzahl von Punkten in das pxyarray diese bei einem entsprechenden Aufruf verbindet:

```
draw_lines()
    {
    pxyarray[0] = 100;
    pxyarray[1] = 100;
    pxyarray[2] = 100;
    pxyarray[3] = 300;
    pxyarray[4] = 500;
    pxyarray[5] = 300;
}
```

### 3.1.1 Plot Point

Was einem BASIC-Anwender da noch bleibt, ist, eine Linie zu zeichnen, deren Anfangs- und Endpunkt identisch sind:

```
linef 1,10,1,10
```

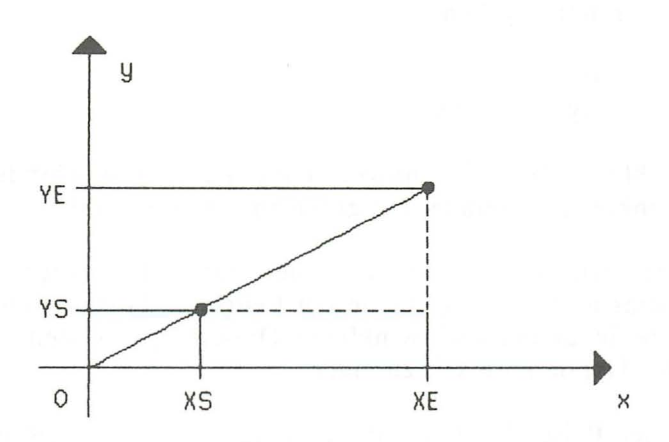
Als C-Programmierer schreibt man sich am besten eine kleine Funktion PLOT(x,y), die bei Bedarf eingebunden werden kann:

```
plot(x,y)
int x,y;
{
    pxyarray[0] = x;
    pxyarray[1] = y;
    v_pmarker(handle,1,pxyarray);
}
```

### 3.1.2 Linien

Linien lassen sich sowohl ausgezeichnet als auch strichliert und punktiert zeichnen. Doch ist dazu immer ein Aufruf von GEM erforderlich. Wem es nicht ganz so auf Schnelligkeit ankommt, der kann auch einen eigenen Algorithmus entwickeln, der ebenfalls in der Lage sein soll, unterbrochene Linien auf den Schirm zu bringen.

Sehen wir uns dazu die folgende Zeichnung an:



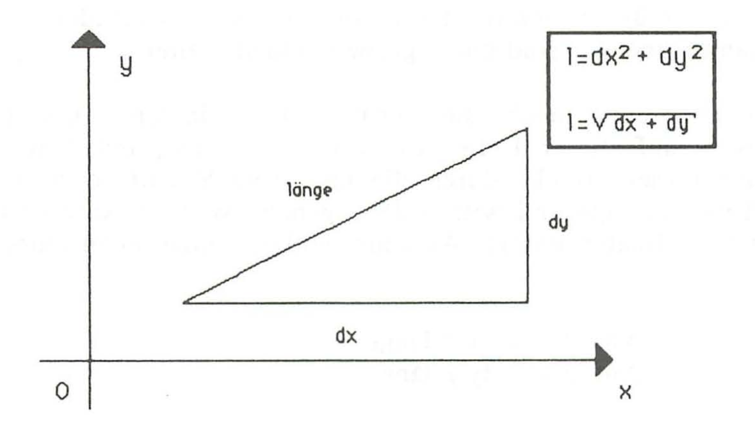




Abb. 17: Zur Berechnung von Geraden

Irgendwo innerhalb des erlaubten Bereiches unserer Zeichenfläche befinden sich zwei Punkte, die wir miteinander verbinden wollen. Alles, was wir von ihnen wissen, sind ihre x- und y-Koordinaten.

Ihren Abstand voneinander können wir zunächst nur für jede Achse getrennt angeben:

dx = xe - xs
dy = ye - ys

Nicht ablesen läßt sich hingegen die kürzeste Verbindung der zwei Punkte, die Gerade, die gezeichnet werden soll.

Erinnern wir uns daher an einen alten Gelehrten namens Pythagoras und machen uns dessen Lehrsatz, daß die Summe der Quadrate in einem rechtwinkligen Dreieck gleich dem Quadrat über der Hypotenuse sei, zu eigen.

Auf unser Beispiel angewandt zeigt sich dann, weil die gesuchte Strecke eben dieser Hypotenuse entspricht, daß wir ihre Länge mit

laenge = SQR(dx * dx + dy * dy)

ansetzen müssen. Somit ist die Anzahl der zu setzenden Punkte bekannt, und wir sind einen großen Schritt weiter.

Um nun auch noch die genauen Koordinaten eines jeden Punktes auf dieser Linie festlegen zu können, teilen wir die Länge dieser Strecke durch die im ersten Schritt festgestellten Differenzbeträge und wissen dann genau, welchen Anteil wir zu den Koordinaten unseres Ausgangspunktes hinzuzählen müssen:

Anteil x = dx / länge
Anteil y = dy / länge

==> x = xs + SCHRITT * dx
y = yb + SCHRITT * dy

SCHRITT ist dabei das gerade zu zeichnende Teilstück (eben genau ein einziger Punkt) der Verbindungslinie und hängt deshalb von der Schrittgröße der Schleifenvariablen ab, die wiederum davon abhängt, wie dicht die Linie gezeichnet werden soll.

Als Ergebnis all dieser Überlegungen können wir nun folgendes Programm formulieren, welches auch leicht in eine andere Sprache umgesetzt werden kann:

```
10    rem draw_line Algorithmus
20    rem --------------------
30    defint a-z
40    clearw 2 : fullw 2
50    input "von welchem x ";xs
60    input "und welchem y ";ys
70    input "nach welchem x ";xe
80    input "und welchem y ";ye
90    input"jeden wievielten Punkt setzen ";sw
100   dx = xe - xs : dy = ye - ys
110   laenge = sqr(dx * dx + dy * dy)
120   for entfernung=0 to laenge step sw
130   x = xs + entfernung * dx / laenge
140   y = ys + entfernung * dy / laenge
150   linef x,y,x,y
160   next punkt
```

### 3.1.3 Kreise

Zwar verfügt das ST-BASIC über die Anweisung circle, doch leider sind die Produkte dieses Aufrufs auch nicht das, was man erwartet. Dies liegt daran, daß der ST lieber ein Vieleck zeichnet, anstatt einen Kreis zu plotten. Meistens ist dieses extrem schnelle Verfahren - denken Sie an das pxyarray - auch ausreichend. Doch wenn es auf mathematische Exaktheit ankommt, kommen wir nicht umhin, uns auch über die mathematische Definition eines Kreises Gedanken zu machen.

So gilt als Kreis der geometrische Ort aller Punkte einer Ebene, die von einem festen Punkt dieser Ebene, dem Kreismittelpunkt, einen konstanten Abstand haben. Weiterhin gilt, daß jede Strecke von diesem Mittelpunkt zu einem Punkt auf der äußersten Begrenzungslinie des Kreises, zur Kreisperipherie, als Radius bezeichnet wird.

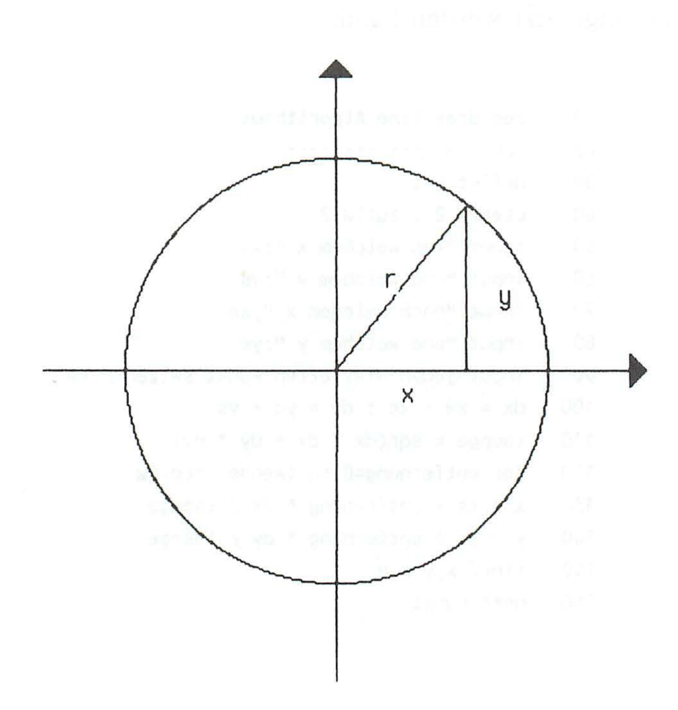



Abb. 18: Zur Definition des Radius

Algebraisch läßt der Radius sich nach dem Satz des Pythagoras mit

r * r = x * x + y * y

r = SQR(x * x + y * y)

angeben, wenn der Mittelpunkt die Koordinaten (0,0) aufweist, die x- und die y-Achse sich also dort schneiden.
Diese Gleichung kann nach y aufgelöst werden:

r * r = x * x + y * y

y = SQR(r * r - x * x)

Um dann einen Kreis zeichnen zu können, müssen wir für x nacheinander alle Werte von -r bis r einsetzen, was sich programmtechnisch bestens durch eine Schleife realisieren läßt:

```
10    rem circle Algorithmus (1)
20    rem ---------------------
30    clearw 2 : fullw 2
50    input "um welchen Mittelpunkt (x/y) ";xm,ym
60    input "Radius ";radius
70    for x=-radius to radius
80    y = sqr(radius * radius - x * x) + ym
90    x1 = x + xm
100   linef x1,y,x1,y
110   linef x1,ym-(y-ym),x1,ym-(y-ym)
120   next x
```

In der Praxis bringt diese Routine jedoch einige Probleme mit sich, denn bei größeren Kreisradien sind die gewählten Schrittweiten zu groß, um noch eine geschlossene Kreislinie zeichnen zu können. Abhilfe schafft ein STEP .1 in Zeile 40, doch wird das Programm dann wegen des zehnfachen Rechnungsaufwandes entsprechend langsamer.

Besser eignet sich da schon ein anderes Verfahren, wenn wir nämlich nicht mehr die Lage eines jeden Punktes in bezug auf die x-Achse berechnen, sondern uns stattdessen 'gradweise' auf der Kreislinie voranbewegen und direkt die x- und y-Koordi-

naten berechnen. Dazu ist jedoch ein kleiner Vorgriff auf folgende Seiten notwendig, denn wir müssen etwas Abstand von unserem gewohnten Koordinatensystem nehmen, und uns den Polarkoordinaten zuwenden. Bei dieser Art der Beschreibung der Lage eines Punktes in einer Ebene wird zum einen der Abstand des Punktes vom Koordinatenursprung, dem Pol, und zum anderen der Winkel zwischen der Abstandsgeraden und der positiven x-Achse angegeben.

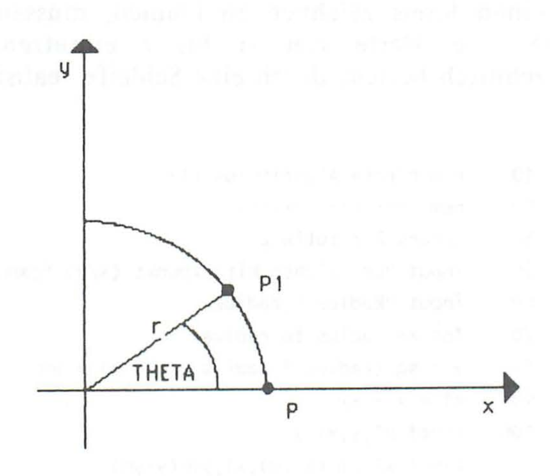



Abb. 19: Polarkoordinaten

Dieses Verfahren ist für unsere Zwecke natürlich ideal. So würden die Angaben für den Punkt p 0 Grad und 50 Längeneinheiten lauten, und um die übrigen Punkte auf der Kreislinie angeben zu können, müßten wir nur die Gradzahlen um jeweils eins erhöhen, bis wir bei 360 Grad angelangt wären.
Und da wir uns x/y -Koordinaten und nicht ominöse Winkelmaße wünschen, wenden wir dabei dann auch gleich zwei sogenannte Transformationsgleichungen an

$$x = r * \cos(theta)$$
$$y = r * \sin(theta)$$

und können auf diese Art und Weise jeden Kreis problemlos zeichnen:

```
10    rem circle Algorithmus(2)
20    rem --------------------
30    clearw 2 : fullw 2
50    input "um welchen Mittelpunkt (x/y) ";xm,ym
60    input "Radius ";radius
70    for grad=1 to 360
80    x = radius * cos(grad) + xm
90    y = radius * sin(grad) + ym
100   linef x,y,x,y
110   next grad
```

Unschön wirkt nur der große Abstand zwischen den einzelnen gesetzten Punkten. Doch ist der Fehler nicht am Algorithmus zu suchen, vielmehr ist unser BASIC dafür verantwortlich, arbeiten sämtliche Winkelfunktionen doch mit Bogenmaß - und unsere Vorgehensweise verlangt Winkelgradmaß.

Eine Anweisung DEG kennt das ST-BASIC bedauerlicherweise nicht, wir werden also selbst für die notwendigen Umrechnungen sorgen müssen.

Nun beträgt der Umfang eines Einheitskreises (ein Kreis mit dem Radius r=1) bekanntlich genau 2 mal pi (da: Kreisumfang = 2*pi*r). Ein Winkel von 360 Grad entspricht demnach 2*pi weshalb wir einen Winkel von 1 Grad in Bogenmaß mit 2*pi/360 angeben können.

Aus diesem Umstand lassen sich problemlos zwei Gleichungen folgern, die uns die Umrechnung von Bogenmaß in Grad und umgekehrt erlauben:

RAD = DEG * pi / 180

DEG = RAD * 180 / pi

Die Praxis beschränkt sich somit auf die Multiplikation mit einem konstanten Faktor c, den wir vor Beginn der eigentlichen Zeichenroutine einmal berechnen (Zeile 40):

```
10    rem circle Algorithmus(3)
20    rem -------------------
30    clearw 2 : fullw 2
40    c = 3.141592 / 180
50    input "um welchen Mittelpunkt (x/y) ";xm,ym
60    input "Radius ";radius
70    for grad=1 to 360
80    x = radius * cos(grad * c) + xm
90    y = radius * sin(grad * c) + ym
100   linef x,y,x,y
110   next grad
```

### 3.1.4 Ellipsen

Sie sind den Kreisen ja recht ähnlich, und so lassen sich die vorgestellten Routinen auch zur Darstellung von Ovalen und Ähnlichem einsetzen.

Um unsere Kreise zu strecken und zu stauchen, reicht es aus, unsere Transformationsgleichungen um einen Faktor zu ergänzen, wie folgende Demonstration zeigt:

```
10    rem ellipse
20    rem -------
30    clearw 2 : fullw 2
40    xm = 320 : ym = 200 : c = 3.141592 / 180
50    radius = 40
60    for grad=1 to 360
70    x = radius * cos(grad * c) * 3 + xm
```

```
80      y = radius * sin(grad * c) + ym
90      linef x,y,x,y
100     next grad
```

VX und VY bewirken hierbei eine Verzerrung des Kreises in der entsprechenden Richtung.

Allerdings sollten Sie bei Ihren Versuchen keine allzu großen Zahlen einsetzen; Werte zwischen null und zehn wären angebracht.

```
10      rem ellipsendemo
20      rem -----------
30      clearw 2 : fullw 2
40      xm = 320 : ym = 200 : c = 3.141592 / 180
50      input "Radius ";radius
60      input "Verzerrung in x oder y-Richtung ";richtung$
70      if richtung$="x" then vx=-1
80      if richtung$="y" then vy=-1
90      for grad=1 to 360
100     for faktor=0.5 to 3.5 step 0.5
110     x=radius * cos(grad * c)
120     if vx then x=x*faktor
130     x = x + xm
140     y=radius * sin(grad * c)
150     if vy then y=y*faktor
160     y = y + ym
170     linef x,y,x,y
180     next faktor
190     next grad
200     for i=1 to 10000 : next
210     end
```

## 3.2 Anwendungen

Auf den folgenden Seiten soll es nun um Anwendungen zweidimensionaler Grafiktechniken gehen. Am bekanntesten sind hier sicherlich Programme zur Erstellung von Geschäftgrafiken, und so will ich Ihnen an dieser Stelle ein komplettes Paket zur Darstellung von Kreis-, Linien- und Balkendiagrammen vorstellen.

### 3.2.1 Kreisdiagramme

Da der Spruch 'Ein Bild sagt mehr als tausend Worte' auch im Geschäftsleben gilt, sind Firmenchefs immer schnell mit eindrucksvollen Grafiken zur Hand, wenn es darum geht, dem Geschäftspartner den eigenen Marktanteil zu verdeutlichen.
Je nach Zusammenhang und Aussageintention der Zahlenstatistik wird eine Darstellungsart gewählt, welche die Verhältnisse ohne längeres Betrachten, Umdenken oder gar Rechnen deutlich macht.

Bewährt haben sich sogenannte Kurvendiagramme, wenn Tendenzen sichtbar gemacht werden sollen.
Als Beispiel mag die Umsatzstatistik einer Firma gelten, bei der die Gesamtumsätze eines jeden Monats für den Zeitraum eines oder mehrere Jahre gegenübergestellt werden. Aus dem Verlauf der Kurve können dann Rückschlüsse auf das Saisongeschäft gewonnen wie auch Vorhersagen für die nächste Zeit gemacht werden.

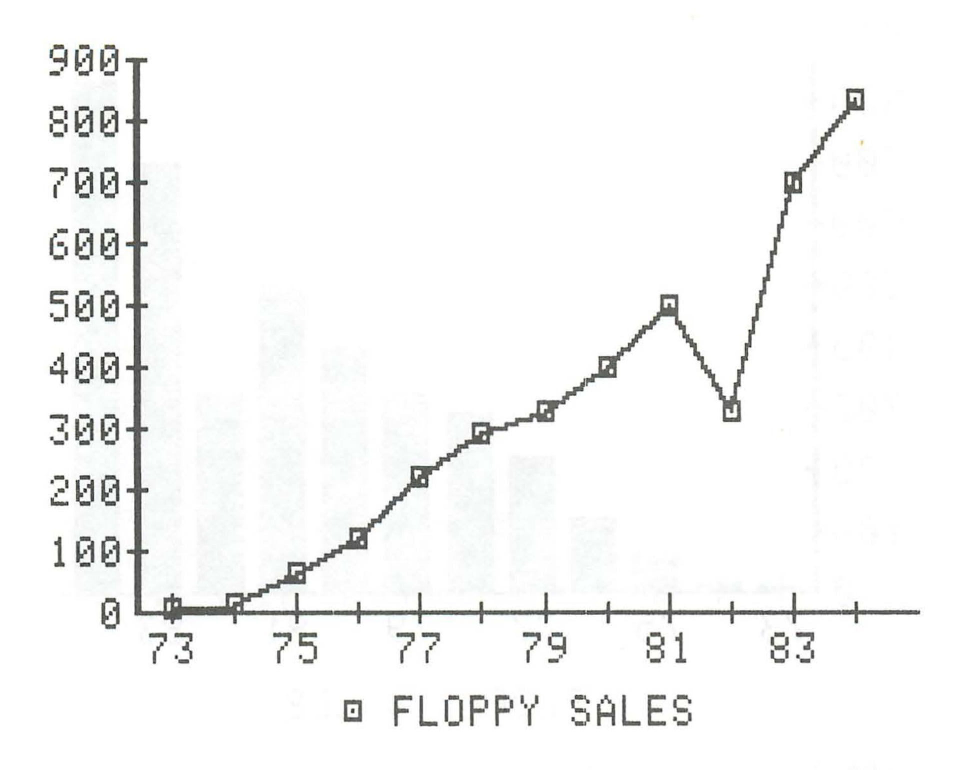



Abb. 20: Liniendiagramm

Balken- und Kreisdiagramme dienen weniger der Interpolation und Trendberechnung, als vielmehr der Gegenüberstellung von einem oder mehreren Datensätzen. Dabei ist die Anzahl der einzelnen Meßwerte natürlich eingeschränkt - eine zehn Meter lange Wand mit fünfhundert Balken wäre zwar eindrucksvoll, aber wenig aussagefähig - weshalb diese Darstellungsart meistens für die Darstellung über größere Zeitspannen gewählt wird. Aber auch hier ist der darzustellende Wert (Umsätze) wieder von einem anderen Faktor (Zeitraum) abhängig.

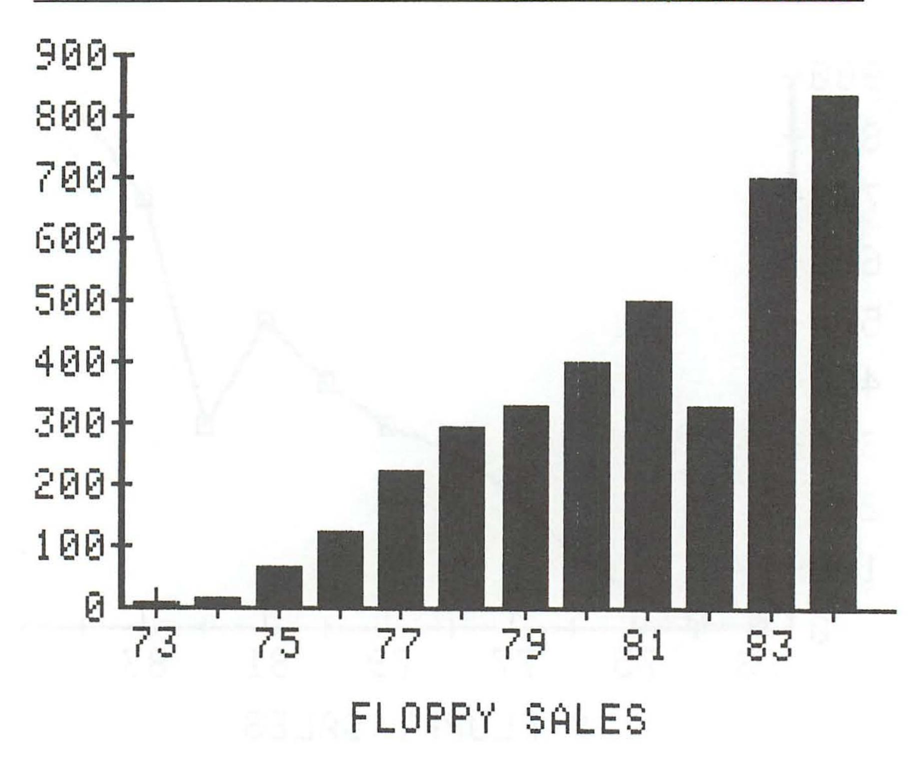



Abb. 21: Balkendiagramm

Ganz anders sieht es bei den Kreis- und Kuchendiagrammen aus. Denn hier ist der größte Wert nicht durch die Zahlenstatistik gegeben, sondern es kann nur eine prozentuale Aufteilung der Kreisfläche stattfinden. Kreisdiagramme eignen sich daher besonders zur Gegenüberstellung von voneinander unabhängigen Werten. Nicht umsonst heißt es, daß 'sich jeder sein Stückchen vom Kuchen abschneiden möchte', und so könnten beispielsweise die Umsatzzahlen von Konkurrenzfirmen Grundlage für diese Darstellung sein.

**Kreisdiagramm**

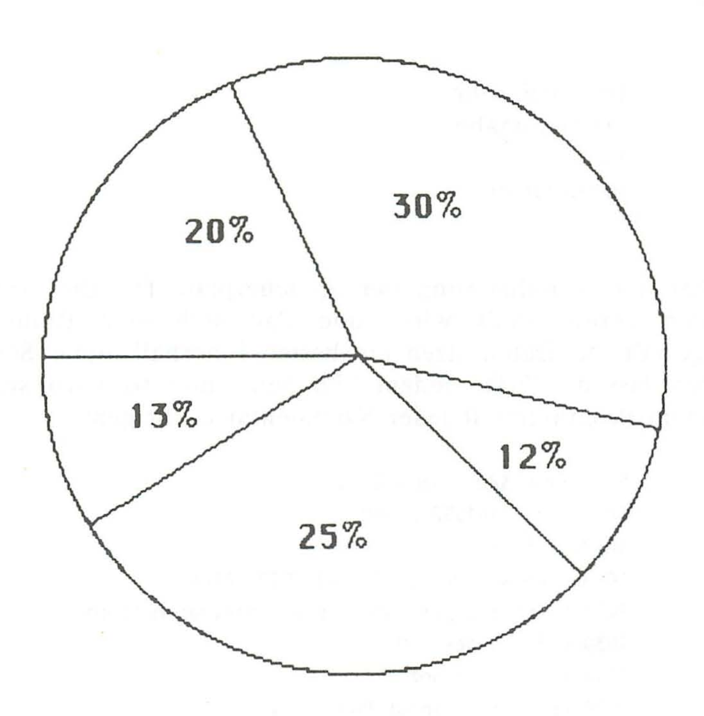



Abb. 22: Kreisdiagramm

Doch welche Schritte werden nötig, um solch ein Kreisdiagramm auf dem Bildschirm entstehen lassen zu können?

Zum einen muß der Kreis erzeugt werden, dann müssen im richtigen Verhältnis zueinander Sektoren abgegrenzt werden und zu guter Letzt sollte auch noch eine Beschriftung der einzelnen Kreissegmente erfolgen, weil die Aussagekraft des Bildes sonst wohl doch recht gering sein dürfte.

Da die Daten auch noch irgendwie an das Programm übermittelt werden müssen, dürfte der grobe Aufbau wohl folgendermaßen aussehen:

Initialisierung
Dateneingabe
Plotten
Beschriften

Daß bei der Initialisierung der Speicherplatz für die Arbeitsvariablen bereitgestellt wird, und daß sich eine Reihe von gleichgestalteten Datensätzen am besten innerhalb einer Schleife eingeben lassen, dürfte jedem klar sein, und so wird sich zu folgendem Programmteil jeder Kommentar erübrigen:

```
50   xm = 320 : ym = 200
60   c = 3.141592 / 180
1000 eingabe:
1010 clearw 2:print "EINGABE DER DATEN"
1020 print : input "Wie viele Einzeldaten ";anz
1030 print : ges = 0
1040 for i=1 to anz
1050 print i; : input "Wert ";wert(i)
1070 ges = ges + wert(i)
1080 print
1090 next i
```

Zur Darstellung von Kreisen sind uns inzwischen ja ausreichend Möglichkeiten bekannt. Da wir uns in diesem Falle jedoch zur Darstellung der einzelnen Kreisabschnitte auf die Kreislinie beziehen müssen, wählen wir die Plotmethode anstelle der schnelleren BASIC-Funktion pcircle.

Um dann die einzelnen Segmente berechnen zu können, bedienen wir uns einer einfachen Dreisatzrechnung bzw. Verhältnisgleichung. Denn die 360 Grad des Vollkreises entsprechen

100 Prozent und somit der Gesamtsumme aller eingegebenen Einzeldaten.

Ein einzelner Eingabewert wird somit soviele Grade von der gesamten Kreislinie in Anspruch nehmen, wie er an Einheiten zur Gesamtsumme beiträgt:

$$\frac{\text{Teilsumme}}{\text{Gesamtsumme}} = \frac{\text{Teilgrad}}{\text{360 Grad}}$$

Somit muß der gesuchte Teilabschnitt (Teilsumme * 360 / Gesamtsumme) Grad betragen:

```
4000 kreisdiagramm:
4020 clearw 2
4030 for i=1 to anz
4040 grad(i) = int(wert(i) * 360/ges)
4050 next i
```

In dem Feld grad(i) sind nun die Kreislängen in Grad eines jeden Teilstückes gespeichert.

Alles, was wir nun noch tun müssen, ist, während des Plottens bei jedem Schritt zu prüfen, ob das Ende eines Sektors erreicht wurde (4100).

Wenn ja, dann wird zum einen eine Gerade zum Kreismittelpunkt gezeichnet, und zum anderen wird die bis zu diesem Zeitpunkt erreichte Gradzahl zum errechneten Endwert des nächsten Teilstückes addiert, so daß gewährleistet ist, daß auch wirklich nur das neue Teilstück berücksichtigt wird:

```
4060 r = 120 : i = 1
4070 for grad=1 to 360
4080 x=r*COS(grad*c)+xm:y=r*SIN(grad*c)+ym : linef x,y,x,y
4100 if grad(i)=grad then linef x,y,xm,ym:grad(i+1)=grad(i+1)+grad(i):i=
i+1
4110 next grad
4120 end
```

Die Beschriftung des Bildes könnte nun immer dann vorgenommen werden, wenn das Zeichnen eines Kreissektors abgeschlossen ist (4100), doch ist zum einen dann die Zuordnung recht zweideutig, und zum anderen sieht es auch nicht gut aus, wenn die Schrift mal ins Bild geht und mal daneben 'steht'.

Wünschenswert wäre die Benennung eines Sektors auf dessen halber Höhe; wir müssen somit bei der Berechnung der Gradanteile eines Sektors auch jeweils die Hälfte dieses Betrages für die Beschriftung gutschreiben:

```
4040 grad(i)=int(wert(i)*360/ges):textpos(i)=int(grad(i)/2)
4060 r = 120 : i = 1 : textalt = 0
4090 IF textalt + textpos(i) = grad then gosub beschriftung
```

Jeweils zum rechten Zeitpunkt kann dann ein Unterprogramm ab Zeile 5000 aufgerufen werden, das zunächst die Länge des aktuellen Titels berechnet. Anschließend wird geprüft, ob die Beschriftung in der linken Hälfte des Kreises erfolgen muß (x<320) oder ob das Label rechtsseitig ausgegeben wird.

Natürlich stellt uns dieses Verfahren dann die Position eines angenommenen Grafikcursors bereit, den wir von BASIC her nur über einen Umweg adressieren können. Zwar wäre auch eine Berechnung der jeweiligen Zeilen- und Spaltenposition möglich, doch wollen wir den exakteren Weg beschreiten und ein universelles Unterprogramm entwickeln, das die Textausgabe an einem beliebigen Koordinatenschnittpunkt erlaubt und das wir auch später bei unseren Plotprogrammen einsetzen können. Wie schon

im vorangegangenem Text erläutert, bedienen wir uns dazu eines Aufrufs des Virtual Device Interfaces.

Die GEMVDI-Funktion v_gtext, Operationscode 8, ermöglicht es Zeichenketten an beliebigen Stellen des Bildschirmes auszugeben (sie ist somit nicht auf das OUTPUT-Fenster beschränkt. Dazu benötigt sie naturgemäß die x- und die y-Koordinate der gewünschten Position, die nach GEM-Konvention in ptsin(0) und ptsin(1) zu übergeben sind.

Der String selbst wird Zeichen für Zeichen im intin-Array abgelegt; contrl(3) muß beim Aufruf des VDI die Anzahl der in intin übergebenen Daten enthalten. Nicht vergessen werden darf, daß Zeichenketten mit einer ASCII-Null abgeschlossen sein müssen. Aus diesem Grunde übergeben wir an contrl(3) die Länge der Zeichenkette + 1:

```
10    rem Aufruf von Gemfunktion 8 : v_gtext
20    rem ---------------------------------
30    text$="Test"
100   poke contrl,8    : rem Funktionsnummer
110   poke contrl+2,1
120   poke contrl+6,len(text$)+1 : rem Laenge Ausgabetext
130   poke ptsin,320 : poke ptsin + 2,200 : rem Koordinaten
140   '
150   rem Text in Array poken
160   for i=1 to len(text$)
170   poke intin + (i-1)*2,asc(mid$(text$,i,1))
180   next i
190   poke intin+(i-1)*2,0 : rem terminating 0
200   vdisys
```

Im Falle unseres Kreisdiagrammes werden wir bei der Positionierung des angenommenen Grafikcursors jeweils genau dessen Relation zum Kreismittelpunkt berücksichtigen müssen.
Sollte beispielsweise der Text links von der Zeichnung stehen, so muß der Cursor um ein Leerzeichen und die Länge der Beschriftung nach links verschoben werden.

Deshalb:

```
5000 beschriftung:
5010 x1 = (len(titel$(i)))
5020 if x>320 then x = x + 32
5030 if x<320 then x = x - x1 * 8 - 40
5040 if y<200 then y = y + 32
5050 if y>200 then y = y + 32
5060 text$ = titel$(i)
5070 gosub v.gtext
```

Allerdings, bevor die Beschriftungen ausgedruckt werden können, müssen Sie diese dem Programm auch mitteilen. Weiterhin müssen Sie es noch um die Subroutine v.gtext ergänzen.

Ein dermaßen vervollständigtes Programm dürfte dann ähnlich dem folgenden Listing aussehen:

```
10    rem kreisdiagramm
20    rem ------------
30    clearw 2 :fullw 2
40    dim wert(20),grad(20)
50    xm = 320 : ym = 200
60    c = 3.141592 / 180
999   '
1000  eingabe:
1010  clearw 2:print "EINGABE DER DATEN"
1020  print : input"Wie viele Einzeldaten ";anz
1030  print : ges = 0
1040  for i=1 to anz
1050  print i; : input"Wert ";wert(i)
1060  input "Bezeichnung ";titel$(i)
1070  ges = ges + wert(i)
1080  print
1090  next i
3999  '
```

```
4000 kreisdiagramm:
4020 clearw 2
4030 for i=1 to anz
4040 grad(i) = int(wert(i) * 360/ges):textpos(i)=int(grad(i)/2 )
4050 next i
4060 r = 120 : i = 1 : textalt = 0
4070 for grad=1 to 360
4080 x=r*cos(grad*c)+xm : y=r*sin(grad*c)+ym : linef x,y,x,y
4090 if textalt + textpos(i) = grad then gosub beschriftung
4095 if grad(i)=grad then textalt = grad
4100 if grad(i)=grad then linef x,y,xm,ym:grad(i+1)=grad(i+1)+grad(i):i
=i+1
4110 next grad
4120 end
4999 '
5000 beschriftung:
5010 x1 = (len(titel$(i)))
5020 if x>320 then x = x + 32
5030 if x<320 then x = x - x1 * 8 - 40
5040 if y<200 then y = y + 32
5050 if y>200 then y = y + 32
5060 text$ = titel$(i)
5070 gosub v.gtext
5080 return
5999 '
6000 v.gtext:
6010 for digit=1 to len(text$)
6020 poke intin + (digit - 1) * 2,asc(mid$(text$,digit,1))
6030 next digit
6040 poke intin + (digit -1 ) * 2,0
6050 poke contrl,8
6060 poke contrl+2,1
6070 poke contrl+6,len(text$)+1
6080 poke ptsin,x
6090 poke ptsin+2,y
6100 vdisys
6110 return
```

### 3.2.2 Balkendiagramme

Rechtecke müssen wir zu Papier bringen, sobald es um Balkendiagramme geht. Viele Hersteller haben den BASIC-Dialekt ihres Computers um einen BOX-Befehl ergänzt, der nach Angabe der linken oberen und der rechten unteren Ecke in Windeseile das gewünschte Rechteck auf den Bildschirm bringt.

Leider haben die Entwickler der ATARI-Systemsoftware darauf verzichtet, was allerdings auch nicht tragisch ist. Vier Linien im rechten Winkel zueinander angebracht, dürften nicht allzu problematisch darzustellen sein; bestenfalls handelt es sich um eine Fleißarbeit, jeweils 4 Koordinatenpaare anzugeben. Doch warum sollte sich jemand mit vielen Koordinaten abmühen, wenn es auch anders geht?

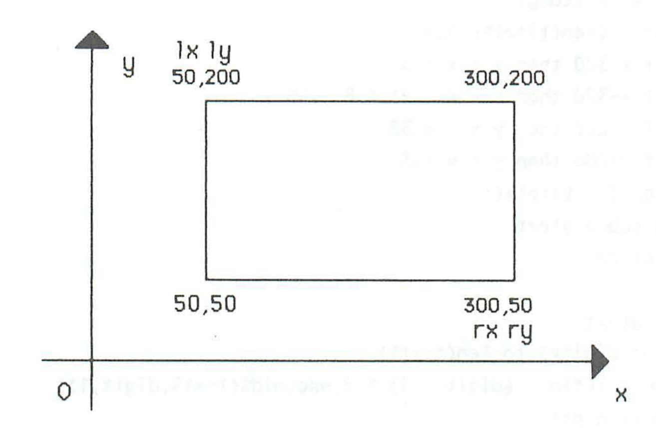



Abb. 23: Zwei Punkte beschreiben ein Rechteck

Wie man sieht, lassen sich die Koordinaten der fehlenden Punkte mit (lx,ry) und (rx,ly) angeben, so daß wir folgendes Unterprogramm schreiben können:

```
10     rem kreisdiagramm
20     rem ------------
30     clearw 2 :fullw 2
40     dim wert(20),grad(20)
50     xm = 320 : ym = 200
60     c = 3.141592 / 180
999    '
1000   eingabe:
1010   clearw 2:print "EINGABE DER DATEN"
1020   print : input"Wie viele Einzeldaten ";anz
1030   print : ges = 0
1040   for i=1 to anz
1050   print i; : input"Wert ";wert(i)
1060   input "Bezeichnung ";titel$(i)
1070   ges = ges + wert(i)
1080   print
1090   next i
3999   '
4000   kreisdiagramm:
4020   clearw 2
4030   for i=1 to anz
4040   grad(i) = int(wert(i) * 360/ges):textpos(i)=int(grad(i)/2 )
4050   next i
4060   r = 120 : i = 1 : textalt = 0
4070   for grad=1 to 360
4080   x=r*cos(grad*c)+xm : y=r*sin(grad*c)+ym : linef x,y,x,y
4090   if textalt + textpos(i) = grad then gosub beschriftung
4095   if grad(i)=grad then textalt = grad
4100   if grad(i)=grad then linef x,y,xm,ym:grad(i+1)=grad(i+1)+grad(i):i
=i+1
4110   next grad
4120   end
4999   '
5000   beschriftung:
5010   x1 = (len(titel$(i)))
5020   if x>320 then x = x + 32
5030   if x<320 then x = x - x1 * 8 - 40
5040   if y<200 then y = y + 32
5050   if y>200 then y = y + 32
5060   text$ = titel$(i)
```

```
5070  gosub v.gtext
5080  return
5999  '
6000  v.gtext:
6010  for digit=1 to len(text$)
6020  poke intin + (digit - 1) * 2,asc(mid$(text$,digit,1))
6030  next digit
6040  poke intin + (digit -1 ) * 2,0
6050  poke contrl,8
6060  poke contrl+2,1
6070  poke contrl+6,len(text$)+1
6080  poke ptsin,x
6090  poke ptsin+2,y
6100  vdisys
6110  return
```

Da GEM und auch der ATARI ST speziell für Grafik ausgelegt worden sind, ist die Geschwindigkeit dabei immer noch so groß, daß optisch kein Unterschied zu einem Computer mit einer entsprechenden Interpreterroutine ausgemacht werden kann. Wie Sie sich selbst überzeugen können, spielt es auch keine Rolle, welchen Punkt Sie zuerst eingeben. Sie müssen nur beachten, daß die Eckpunkte sich diagonal gegenüberstehen.

Auch eine FILLED BOX, ein ausgezeichnetes Rechteck, kann wie gehabt auf den Bildschirm gebracht werden:

```
10    clearw 2 : fullw 2
20    input "links oben (x,y) ";lx,ly
30    input "rechts unten (x,y) ";rx,ry
40    linef lx,ly,rx,ly
50    linef rx,ly,rx,ry
60    linef rx,ry,lx,ry
70    linef lx,ry,lx,ly
80    goto 20
```

Durch Verändern der Schrittweite in Zeile 130 können wir so auch unterschiedlich gerasterte Flächen erzeugen. Allerdings weist diese Vorgehensweise noch einen Nachteile auf: Denn bedingt durch die Schleifenkonstruktion müssen die Eckpunkte immer in der Reihenfolge links/rechts angegeben werden.

Glücklicherweise kennt das ST-BASIC jedoch einen Befehl FILL, der es erlaubt, beliebige Flächen mit einem Füllmuster auszufüllen. Als Parameter müssen wir bei dessen Aufruf nur einen Punkt innerhalb der zu füllenden Fläche angeben. Diesen können wir anhand unserer Eckkoordinaten leicht bestimmen:

```
10     clearw 2 : fullw 2
20     input "ECKPUNKT 1 (x,y) ";lx,ly
30     input "ECKPUNKT 2 (x,y) ";rx,ry
40     linef lx,ly,rx,ly
50     linef rx,ly,rx,ry
60     linef rx,ry,lx,ry
70     linef lx,ry,lx,ly
80     mx = (lx + rx) / 2
90     my = (ly + ry) / 2
100    fill mx,my
110    goto 20
```

Es ist nun an der Zeit, unser Programm Pieplot um die Möglichkeit zur Darstellung von Balkendiagrammen, sogenannten Bar-Charts, zu erweitern. Dabei wollen wir eine Darstellung gemäß folgender Abbildung anstreben:

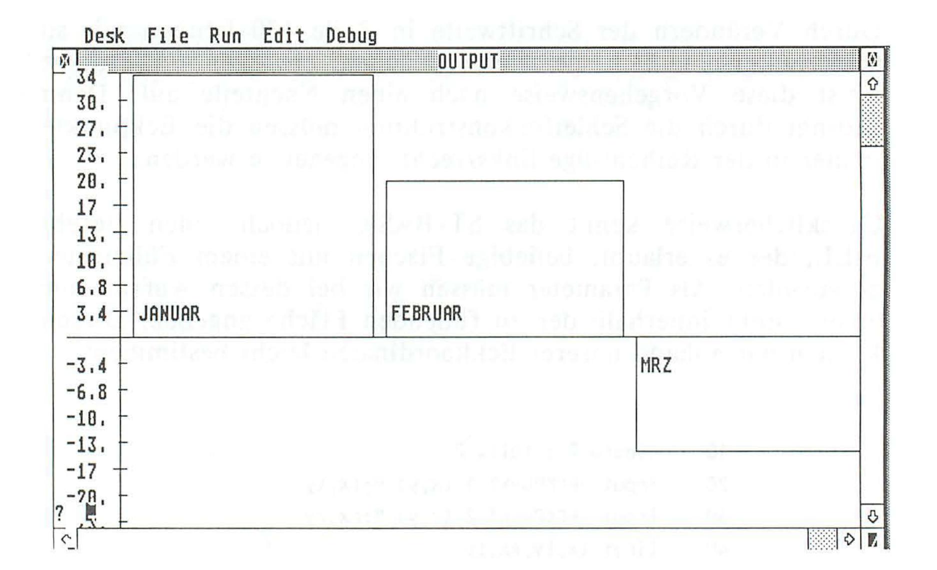



Abb. 24: Hardcopy zu BARCHART.BAS

Prinzipiell stellen sich uns dabei die gleichen Probleme wie schon zuvor, denn die Daten müssen wiederum erfaßt und vor dem eigentlichen Zeichnen an die Zeichenfläche angepaßt, somit also skaliert werden.

Diese Skalierung ist einer der wichtigsten Arbeitsvorgänge, der bei grafischen Darstellungen gleich welcher Art berücksichtigt werden muß. Aus diesem Grunde soll das Problem der Maßstabsanpassung an dieser Stelle etwas ausführlicher erläutert werden.

Was tatsächlich zu geschehen hat, ist immer ein Vergleich zwischen zwei Größen. So werden einerseits die Datensätze nach einer oberen Grenze ausgerichtet, wie es beispielsweise bei den Kreisdiagrammen der Fall war (obere Grenze gleich 360 Grad). Oder aber es wird der größte darzustellende Meßwert als Bezugsgröße eingeführt - der dann in Relation zu den physikalisch vorhandenen Punkten der Darstellung gesetzt wird - und alle übrigen Daten werden nach diesem Maximum ausgerichtet.

Dies kommt auch dem Wunsch nach einer möglichst großen Darstellung auf der nutzbaren Zeichenfläche entgegen. Als Skalierungsfaktor können wir somit den Quotienten aus der größtmöglichen Ausdehnung der Zeichenfläche und dem Maximalwert der Daten angeben:

$$SK = \frac{PLOTMAX}{DATAMAX}$$

Durch Multiplikation eines jeden Datenwertes mit diesem Skalierungsfaktor werden wir die abzubildende Meßreihe an die Darstellung des größten Wertes, und somit an die Zeichenfläche, anpassen. Das Produkt aus Datenwert * Skalierungsfaktor kann somit bereits als direkte Entfernungsangabe des abzubildenden Wertes in Bezug auf die Nullinie der Zeichenfläche aufgefaßt werden.

Da die Koordinaten dieser Grundlinie zu Beginn des Zeichenvorganges gewählt, die darzustellende Entfernung von dieser Grundlinie berechnet wird und zur Darstellung eines Rechteckes erwiesenermaßen nur zwei Punkte bekannt sein müssen, ist unser Problem bereits gelöst, und das zuvor erstellte Programm kann entsprechend abgeändert werden:

```
40   dim wert(20),grad(20),bwert(20)
50   ym = 200
1000 eingabe:
1030 print : ges = 0 : wmax = 0
```

Diese Zeilen stellen zunächst einmal den benötigten Variablenspeicherplatz bereit.

In Zeile 1050 wird eine Ergänzung notwendig, welche den größten Eingabewert als wmax notiert, so daß im folgenden der Skalierungsfaktor wie erläutert berechnet werden kann:

```
1070 ges = ges + wert(i) : if wert(i) > wert(i-1) then wmax = wert(i)
```

```
2010 clearw 2 : mfaktor = 199 / wmax : rx = 60 : abstand =10
```

Vielleicht wundert es Sie nun, warum bei der Berechnung des Skalierungsfaktors, der hier als Maßstabsfaktor bezeichnet wird, nur eine maximale Ausdehnung der Zeichenfläche von 199 und nicht 400 (Punkte in y-Richtung) angegeben werden.
Sie sollten jedoch bedenken, daß zum einen in solchen Diagrammen auch die Möglichkeit zur Darstellung negativer Werte gewahrt bleiben sollte, weshalb der Maximalwert sich halbiert, und zum anderen könnte es geschehen, daß die obere Begrenzungslinie nicht mehr gezeichnet wird, weil sich durch die Multiplikation mit mf ein etwas größerer Wert als 200 ergibt. Im Sinne einer späteren Beschriftung der y-Achse wäre es sogar sinnvoll, die obere Grenze noch etwas niedriger anzusetzen.

Unter Beachtung dieser Punkte kann die x-Achse somit nur in der Mitte des Schirmes liegen; die y-Achse wird ein klein wenig von der linken Zeichenflächengrenze abgesetzt, so daß noch genügend Platz für eine spätere Beschriftung bleibt:

```
2020 linef 10,200,640,200 : REM x-achse
2030 linef 50,0,50,400    : REM y-Achse
```

Bevor die einzelnen Balken dann gezeichnet werden, muß innerhalb einer Schleife deren Länge berechnet werden, außerdem wird ihre Breite in Abhängigkeit von ihrer Anzahl, der zur Verfügung stehenden Bildfläche und dem zwischen ihnen liegenden Abstand:

```
2040 for i=1 to anz
2050 bwert(i) = wert(i) * mfaktor
2060 next i
```

```
2070 balkenbreite = 580 / anz - abstand
```

Die Balken selbst werden dann mit einer der zuvor entwickelten Routinen dargestellt:

```
2080 for i=1 to anz
2090 ly = 200 : lx = rx + balkenbreite : ry = 200 - bwert(i)
2130 linef lx,ly,rx,ly : linef rx,ry,lx,ry
2140 linef lx,ry,lx,ry : linef lx,ry,lx,ly
2150 rx = rx + abstand + balkenbreite:REM Beginn des naechsten Balkens
2160 next i
```

Was wir dann noch zu erledigen hätten, wäre eine Beschriftung der Achsen. Für die waagerechte x-Achse ist diese Prozedur nicht weiter problematisch, doch ehe wir die y-Achse ebenfalls korrekt beschriften können, werden wir doch etwas zusätzlichen Aufwand treiben müssen.

Denn da nicht nur einfach gleichmäßige Abstände markiert werden sollen, sondern diese entsprechend der dargestellten Daten beschriftet werden müssen, ergibt sich daraus schon eine Multiplikation mit dem Skalierungsfaktor.

Dabei ist der Wert des größten dargestellten Punktes mit wmax identisch, weshalb ein Teilstück so groß wie wmax dividiert durch die gewünschte Anzahl dieser Teilstücke, hier zehn, sein muß:

```
2170 REM Skalierung y-Achse
2180 x = 10 : for y1=1 to 10
2190 linef 55,200 + y1 * 20,45,200 + y1 * 20
2200 y = 260 + y1 * 20 - 15:y2 = wmax / 10 * y1 *-1
2210 text$ = left$(str$(y2),4)
2220 gosub v.gtext
2230 next y1
```

Entsprechend kann auch der negative Teil der y-Achse beschriftet werden:

```
2240 for y1=1 to 10
2250 linef 55,y1 * 20,45,y1 * 20
2260 y = 260 - y1 * 20 - 15:y2 = wmax / 10 * y1
2270 text$ = left$(str$(y2),4)
2280 gosub v.gtext
2290 next y1
```

Eine Beschriftung der x-Achse kann folgendermaßen vorgenommen werden:

```
2100 x = rx + 5 : y = 245 : if ry>200 then y = y+20
2110 if ry < 200 then y = y - 20
2120 text$ = titel$(i) : gosub v.gtext
```

Ihnen steht nach Eingabe dieser Zeilen nun ein Programm zur Verfügung, das in der Lage ist, eine Reihe von geeigneten Eingabedaten Balkendiagramm darzustellen. Dabei werden sämtliche Skalierungen, auch eine Anpassung der Werte zur formatfüllenden Darstellung, automatisch vorgenommen.

Allerdings sollten Sie beachten, daß auch dem ATARI ST und dem hochauflösendem SW-Monitor physikalische Grenzen gesetzt sind, die auch kein noch so gutes Programm umgehen kann. Versuchen Sie daher bitte nicht, allzu viele Einzeldaten innerhalb einer einzigen Grafik darzustellen.

```
10    rem balkendiagramm
20    rem --------------
30    clearw 2 :fullw 2
40    dim wert(20),bwert(20)
50    ym = 200
```

```
999    '
1000   eingabe:
1010   clearw 2:print "EINGABE DER DATEN"
1020   print : input"Wie viele Einzeldaten ";anz
1030   print : ges = 0 : wmax = 0
1040   for i=1 to anz
1050   print i; : input"Wert ";wert(i)
1060   input "Bezeichnung ";titel$(i)
1070   ges = ges + wert(i) : if wert(i) > wert(i-1) then wmax = wert(i)
1080   print
1090   next i
2000   barchart:
2010   clearw 2 : mfaktor = 199 / wmax : rx = 60 : abstand = 10
2020   linef 10,200,640,200 : rem x-Achse
2030   linef 50,0,50,400    : rem y-Achse
2040   for i=1 to anz
2050   bwert(i) = wert(i) * mfaktor
2060   next i
2070   balkenbreite = 580 / anz - abstand
2080   for i=1 to anz
2090   ly=200 : lx = rx + balkenbreite : ry = 200 - bwert(i)
2100   x = rx + 5 : y = 245 : if ry > 200 then y = y + 20
2110   if ry < 200 then y = y - 20
2120   text$ = titel$(i) : gosub v.gtext
2130   linef lx,ly,rx,ly : linef rx,ly,rx,ry
2140   linef rx,ry,lx,ry : linef lx,ry,lx,ly
2150   rx = rx + abstand + balkenbreite
2160   next i
2170   ' Beschriftung y-Achse
2180   x = 10 : for y1 = 1 to 10
2190   linef 55,200 + y1 * 20,45,200 + y1 * 20
2200   y = 260 + y1 * 20 - 15 : y2 = wmax / 10 * y1 * -1
2210   text$ = left$(str$(y2),4)
2220   gosub v.gtext
2230   next y1
2240   for y1 = 1 to 10
2250   linef 55,y1 * 20,45,y1 * 20
2260   y = 260 - y1 * 20 -15 : y2 = wmax / 10 * y1
2270   text$ = left$(str$(y2),4)
2280   gosub v.gtext
```

```
2290  next y1
2300  input e$
2310  end
4999  '
5999  '
6000  v.gtext:
6010  for digit=1 to len(text$)
6020  poke intin + (digit - 1) * 2,asc(mid$(text$,digit,1))
6030  next digit
6040  poke intin + (digit -1 ) * 2,0
6050  poke contrl,8
6060  poke contrl+2,1
6070  poke contrl+6,len(text$)+1
6080  poke ptsin,x
6090  poke ptsin+2,y
6100  vdisys
6110  return
```

### 3.3 Manipulation von 2-D-Bildern

Bislang haben wir bei der Erstellung unserer Grafiken nur die grundsätzlichen Grafikbefehle des ATARI ST genutzt. Wie das vorangegangene Programm zeigte, läßt sich damit schon eine Menge erreichen. Allein durch das Setzen von Punkten ließen sich wohl sämtliche Aufgaben der grafischen Datenverarbeitung realisieren, doch wäre die Erstellung von Bildern dann wohl ein ebenso mühseliges wie auch zeitaufwendiges Unterfangen. Und der Versuch einer Programmierung von Anwendungsprogrammen, welche die so erstellten Grafiken manipulieren, wäre von vornherein zum Scheitern verurteilt.

Man muß sich daher nach geeigneten Hilfsmitteln umsehen und findet diese dann auch im Bereich der Mathematik. Hier lassen sich Formeln finden, die gewissermaßen als Patentrezepte angesehen werden können, und in die nur die entsprechenden Zahlenwerte eingesetzt werden müssen.

Im Rahmen dieses letzten Teils des dritten Kapitels sollen daher einige mathematische Grundlagen vorgestellt und durch Beispielprogramme belegt werden. Dabei werden wir uns Schritt für Schritt auch dem Ziel, das wohl ein jeder Besitzer eines grafikfähigen Computers hat, nämlich der dreidimensionalen Bildverarbeitung, annähern.

### 3.3.1 Shapes

Sie lassen sich in den verschiedensten Heim- und Hobbycomputern in Form einer entsprechenden Anweisung finden. Nicht jedoch bei unserem ST!

Doch ist dies kein Grund zur Traurigkeit, werden diese Figuren doch immer softwaremäßig realisiert - und Speicherplatz ist schließlich ausreichend vorhanden, auch die Arbeitsgeschwindigkeit des 68000-Prozessors wird wohl hoch genug sein, weshalb wir im folgenden zunächst die Technik der Shapes unter Nutzung der vorhandenen Plotbefehle realisieren wollen.

Im Sinne des Erfinders ist ein Shape die Beschreibung einer Figur durch deren Eckpunkte, die dann durch Linien miteinander verbunden werden. Der äußere Umriß des hier abgebildeten Hauses kann also mit

250,260
350,260
350,180
250,180
350,180
250,180

beschrieben werden.

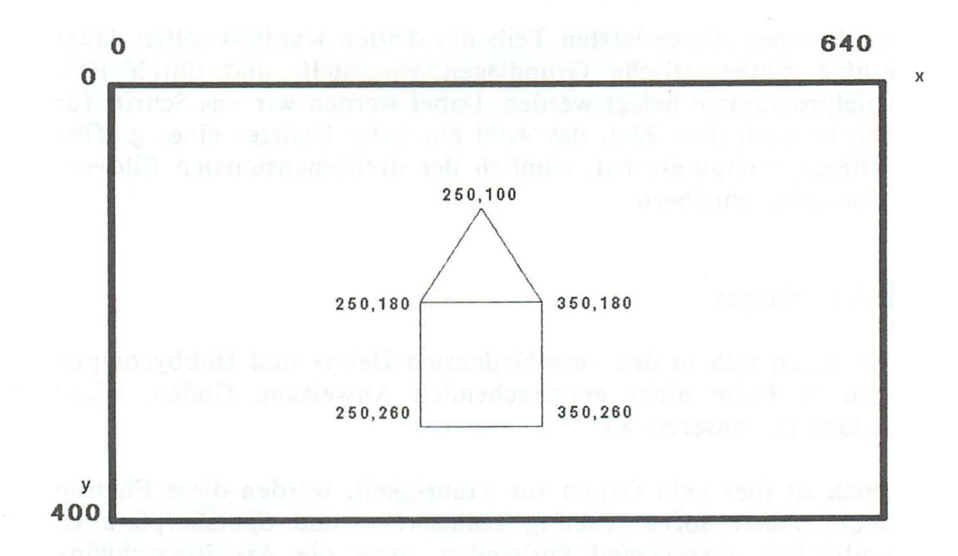



Abb. 25: Die Beschreibung eines Hauses in RC

Um das Bild des Hauses auf den Schirm zu bringen, werden die meisten Computerbesitzer wohl ein Programm wie das hier abgebildete benutzen:

```
10    fullw 2:clearw 2
20    linef 250,260,350,260 REM oben
30    linef 350,260,350,180 REM rechts
40    linef 350,180,250,180 REM unten
50    linef 250,180,250,260 REM links
60    linef 350,180,300,100 REM dach r
70    linef 300,100,250,180 REM dach l
```

Eine alternative Methode bewahrt die Bilddaten innerhalb eines Blockes auf, anstatt sie über mehrere Anweisungszeilen zu verteilen. Die einzelnen Linien werden dann auch nicht mehr direkt, sondern von einem speziellen Programmteil gezeichnet:

```
10      fullw 2:clearw 2
20      read xv,yv REM Anfangspunkt
30      read xn,yn REM Endpunkt
40      linef xv,yv,xn,yn REM Linie zeichnen
50      goto 20     REM und weiter
1000    REM Nachfolgend die Bilddaten:
1020    data 250,260,350,260,350,260,350,180,350,180,250,180
1030    data 250,180,250,260,250,180,300,100,300,100,350,180
```

Noch bricht das Programm mit einer Fehlermeldung ab, da ihm die Anzahl der Eckpunkte nicht bekannt war, doch läßt sich dieser Umstand, zumal es sich bei Shapes fast immer um vordefinierte Figuren handelt, schnell beheben.

```
10      fullw 2:clearw 2
20      read anz   REM Anzahl Linien
30      for linie =1 to anz
40      read xv,yv REM Anfangspunkt
50      read xn,yn REM Endpunkt
60      linef xv,yv,xn,yn REM Linie zeichnen
70      next linie REM und weiter
80      end
1000    REM Nachfolgend die Bilddaten:
1010    data 6
1020    data 250,260,350,260,350,260,350,180,350,180,250,180
1030    data 250,180,250,260,250,180,300,100,300,100,350,180
```

Das erste Element des Datenblockes gibt an, aus wie vielen Elementen das Shape existiert, denn um nichts anderes als um die Shapedaten handelt es sich bei den Programmzeilen von 1010 bis 1030.

Vielleicht erscheint Ihnen diese Konstruktionsweise für das einfache Zeichnen irgendwelcher Figuren zu aufwendig, doch beachten Sie, wie leicht sich die Bilder austauschen lassen:

```
10      fullw 2:clearw 2
20      read anz   REM Anzahl Linien
25      if anz<0 then end
30      for linie =1 to anz
40      read xv,yv REM Anfangspunkt
50      read xn,yn REM Endpunkt
60      linef xv,yv,xn,yn REM Linie zeichnen
70      next linie REM und weiter
80      end
999     '
1000    REM Nachfolgend die Bilddaten:
1010    data 7
1020    data 300,350,450,350,450,350,400,300
1030    data 400,300,400,200,400,200,375,150
1040    data 375,150,350,200,350,200,350,300
1050    data 350,300,300,350
1060    data -1
```

Ebenso ist das Programm nach einer kleinen Änderung in der Lage, beliebig viele Shapes - und somit komplette Bilder zu erzeugen. Wir müssen nur ein Kennzeichen vereinbaren, das das Datenende vereinbart.

Im Falle des ST-BASIC, wo der Koordinatenursprung immer bei 0,0 liegt, kann eine negative Zahl gewählt werden. Sind von der Konstruktion des Koordinatensystemes diese als gültige Angaben erlaubt (Logo), ist ein Wert zu wählen, der außerhalb der Zeichenfläche liegt.

```
5       rem haus4
10      fullw 2:clearw 2
20      read anz   REM Anzahl Linien
25      if anz<0 then end
30      for linie =1 to anz
40      read xv,yv REM Anfangspunkt
50      read xn,yn REM Endpunkt
60      linef xv,yv,xn,yn REM Linie zeichnen
70      next linie REM und weiter
```

```
80      goto 20
1000    REM Nachfolgend die Bilddaten:
1010    data 6
1020    data 250,260,350,260,350,260,350,180,350,180,250,180
1030    data 250,180,250,260,250,180,300,100,300,100,350,180
1040    data 3
1050    data 280,260,280,220,280,220,320,220,320,220,320,260
1060    data 4
1070    data 260,210,290,210,290,210,290,190,290,190,260,190,260,190,260,2
10
1080    data 4
1090    data 310,210,340,210,340,210,340,190,340,190,310,190,310,190,310,2
10
1100    data -1
```

### 3.3.2 Koordinatentransformation

Nun sollten Sie das vorherige Programm folgendermaßen ändern:

```
5       rem trans1
10      fullw 2:clearw 2
15      for faktor=0.1 to 1.9 step 0.2
16      restore
20      read anz
25      if anz<0 then 80
30      for linie =1 to anz
40      read xv,yv
50      read xn,yn
60      linef xv*faktor,yv*faktor,xn*faktor,yn*faktor
70      next linie
80      next faktor
90      end
1000    REM Nachfolgend die Bilddaten:
1010    data 17
1020    data 250,260,350,260,350,260,350,180,350,180,250,180
1030    data 250,180,250,260,250,180,300,100,300,100,350,180
1050    data 280,260,280,220,280,220,320,220,320,220,320,260
```

```
1070 data 260,210,290,210,290,210,290,190,290,190,260,190,260,190,260,2
10
1090 data 310,210,340,210,340,210,340,190,340,190,310,190,310,190,310,2
10
1100 data -1
```

Wie Sie sehen, erhalten Sie gleich eine Reihe von Häusern, wobei eins größer als das andere ist.

Die Skalierung, das Vergrößern und Verkleinern von Objekten, läßt sich also auf einfache Weise durch Multiplikation mit einem konstanten Faktor bewerkstelligen. Mathematisch betrachtet, handelt es sich bei diesem Vorgang um eine Koordinatentransformation, die anhand von Matrizen dargestellt werden kann.

Dadurch eröffnen sich dem Anwender noch eine Reihe von weiteren Möglichkeiten, die wir nun im einzelnen untersuchen wollen.

#### 3.3.2.1 Das Koordinatensystem

Verschiebungen, Vergrößerungen, Verkleinerungen und auch Spiegelungen lassen sich im zweidimensionalen Fall ohne allzu aufwendige Berechnungen im kartesischen Koordinatensystem bewerkstelligen. Um jedoch die Auswirkungen der verschiedensten Manipulationen auf dem Bildschirm beobachten zu können, ist es nicht mehr ausreichend, nur innerhalb des ersten Quadranten zu arbeiten. Aus diesem Grunde legen wir für alle folgenden Programme den Koordiantenursprung in der Mitte des Bildschirmes fest und zeichnen als Orientierungshilfe auch die beiden Achsen ein.

Dabei taucht zunächst die Frage auf, wie wir den Ursprung unseres Systems in die Mitte legen, schließlich kennt BASIC keinen entsprechenden Befehl. Nun, erinnern wir uns der Ausführungen zur Wertanpassung der Eingabedaten bei unserem Businessgrafikprogramm. Auf ganz ähnliche Art und Weise gehen wir auch in diesem Falle vor, um ein Koordinatensystem

mit dem Wertebereich von -320 bis +320, bzw. -200 bis +200 zu erlangen, bei dem der Ursprung dann in Schirmmitte liegt. So wie wir bei Barplot die Werte PLOTMAX und DATAMAX in Relation zueinander gesetzt haben, so ist es hier nun die Länge der Strecke des abzubildenden Punktes in x- bzw. y-Richtung der tatsächlich vorhandenen (X, Y) zu den gewünschten Koordinaten (XMIN, XMAX, YMIN, YMAX).

$$\text{Für x gilt: } \frac{(X - XMIN)}{(XMAX - XMIN)}$$

$$\text{Für y gilt: } \frac{(YMAX - Y)}{(YMAX - YMIN)}$$

Natürlich müssen sowohl der x- als auch der y-Wert noch mit dem Maß der vorhandenen Auflösung multipliziert werden, schließlich handelt es sich nur um eine relative Angabe, die mit dem Faktor der Maximalausdehnung (640 bzw. 400) auf das wirklich vorhandene Maß gebracht werden muß.

Weiterhin müssen für XMIN, XMAX, YMIN und YMAX in obiger Formel die gewünschten Parameter eingesetzt werden, d.h., für das gewünschte Koordinatensystem betragen die Werte:

```
20 xmin = -320 : xmax =  320
30 ymin = -200 : ymax =  200
```

Somit lassen sich als Transformationsgleichung folgende Zeilen angeben:

$$X = 640 * (X - XMIN) / (XMAX - XMIN)$$
$$Y = 400 * (YMAX - Y) / (YMAX - YMIN)$$

Für unsere Zwecke formulieren wir zwei kurze Unterprogramme, von denen eines die Lage eines Punktes und das andere die Lage von zwei Punkten neu berechnet. Aufrufen lassen sollen sie sich mit GOSUB TRANSPOINT und GOSUB TRANSLINE:

```
10000 TRANSPOINT:
10010 X = 640 * (X - XMIN) / (XMAX - XMIN)
10020 Y = 400 * (YMAX - Y) / (YMAX - YMIN)
10030 RETURN

10050 TRANSLINE:
10060 XV = 640 * (XV - XMIN) / (XMAX - XMIN)
10070 YV = 400 * (YMAX - YV) / (YMAX - YMIN)
10080 XN = 640 * (XN - XMIN) / (XMAX - XMIN)
10090 YN = 400 * (YMAX - YN) / (YMAX - YMIN)
10100 RETURN
```

Problemlos läßt sich nun das gewünschte Koordinatensystem auf den Schirm bringen:

```
5     rem quadranten
10    clearw 2:fullw 2
20    xmin = -320 : xmax = 320
30    ymin = -200 : ymax = 200
110   xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
120   linef xv,yv,xn,yn
130   xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
140   linef xv,yv,xn,yn
150   end
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
10050 transline:
```

```
10060 xv = 640 * (xv - xmin) / (xmax - xmin)
10070 yv = 400 * (ymax - yv) / (ymax - ymin)
10080 xn = 640 * (xn - xmin) / (xmax - xmin)
10090 yn = 400 * (ymax - yn) / (ymax - ymin)
10100 return
```

Berechnet werden die Achsen durch den Aufruf unseres Unterprogrammes in den Zeilen 110 und 130.

### 3.3.2.2 Verschiebungen

Die Parallelverschiebung ist die einfachste Art der Transformation eines kartesischen Koordinatensystemes. Sie besteht aus einer Verschiebung der Achsen ohne deren Richtungen zu ändern.

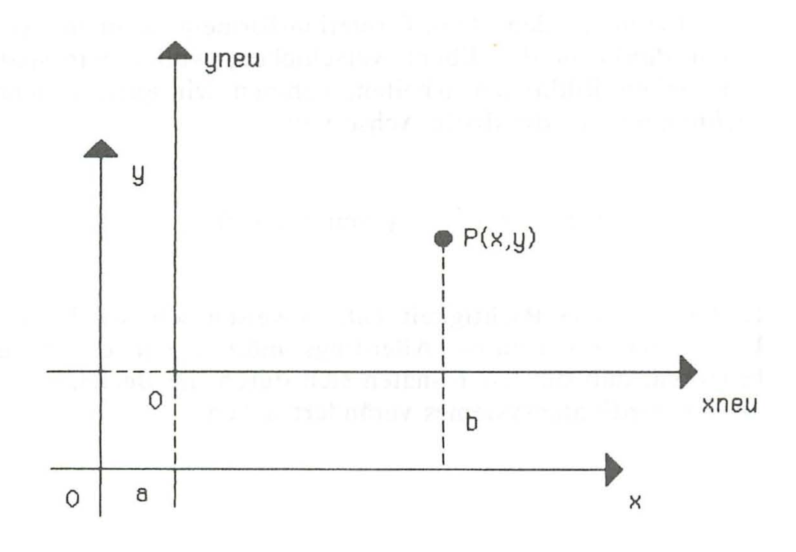



Abb. 26: Zur Definition der Parallelverschiebung

Der Skizze läßt sich leicht entnehmen, daß die Größe der Verschiebung durch die Summanden a und b festgelegt wird.

Der Zusammenhang zwischen dem alten und dem neuen Koordinatensystem läßt sich also wie folgt formulieren:

xneu = x - a

yneu = y - b

Da wir aber nicht die Achsen, sondern den Punkt verschieben wollen, lauten die für uns richtigen Gleichungen:

xneu = x + a

yneu = y + b

Mit diesen beiden Transformationsformeln können wir dann jeden Punkt in der Ebene verschieben; wollen wir später mit räumlichen Bildnissen arbeiten, nehmen wir entsprechende Berechnungen für die dritte Achse vor:

zneu = z + c (zneu = z - c)

Daß alles seine Richtigkeit hat, beweisen wir wieder mit dem Umriß unseres Hauses. Allerdings müssen wir dabei zunächst beachten, daß die Koordinaten sich durch die Beziehung auf das neue Koordinatensystemes verändert haben.

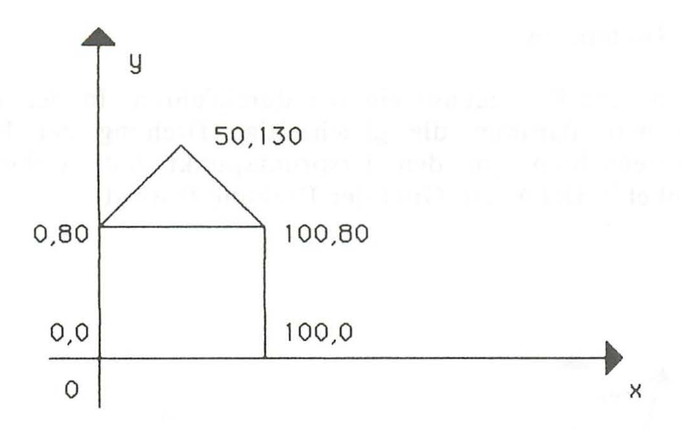



Abb. 27: Parallelverschiebung

```
5     rem quadranten
10    clearw 2:fullw 2
20    xmin = -320 : xmax = 320
30    ymin = -200 : ymax = 200
110   xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
120   linef xv,yv,xn,yn
130   xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
140   linef xv,yv,xn,yn
150   end
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
10050 transline:
10060 xv = 640 * (xv - xmin) / (xmax - xmin)
10070 yv = 400 * (ymax - yv) / (ymax - ymin)
10080 xn = 640 * (xn - xmin) / (xmax - xmin)
10090 yn = 400 * (ymax - yn) / (ymax - ymin)
10100 return
```

### 3.3.2.3 Drehungen

Sie lassen sich fast ebenso einfach durchführen. In der Ebene versteht man darunter die gleichzeitige Drehung der beiden Koordinatenachsen um den Ursprungspunkt 0,0, wobei der Drehwinkel THETA den Grad der Drehung festlegt.

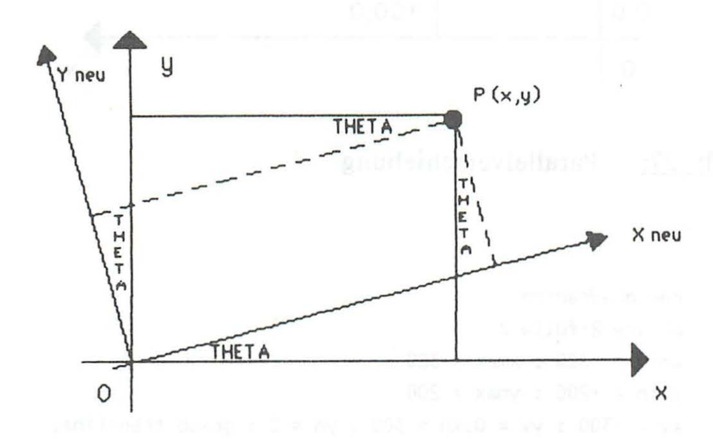



Abb. 28: Drehung

Wenn Sie die Ausführungen zur Konstruktion von Kreisen gelesen haben, so werden Sie rasch die Beziehung zwischen den Punkten x und xneu, wie auch y und yneu feststellen:

$$xneu = x * \cos(theta) + y * \sin(theta)$$

$$yneu = -x * \sin(theta) + y * \cos(theta)$$

### 3.3.3 Matrizen

Bei der Drehung wie auch bei der Verschiebung handelt es sich um sogenannte lineare Transformationen. Diese werden gerne in sogenannter Matrizenform, in Zahlentafeln, dargestellt. Dabei erfolgt die Anordnung der Zahlen, die nun als Matrixelemente bezeichnet werden, in Reihen und Spalten. Jede Zahl der Matrix kann dabei über ihre Position innerhalb des Schemas angesprochen werden, wobei die Indizierung nach dem gleichen Schema wie bei den Arrays in BASIC erfolgt - nur auf das Komma wird verzichtet.

```
      [a11  a12  a13]

M =   [a21  a22  a23]

      [a31  a32  a33]
```

Die Matrix selbst ist ein reines Zahlenschema, man kann nur die Anordnung der einzelnen Elemente angeben, nicht aber einen Gesamtwert berechnen. Dennoch, findige Mathematiker haben nicht eher geruht, bis ganze Regale in Universitätsbibliotheken mit Rechenregeln für Matrizen gefüllt waren. Selbst nur die wichtigsten Regeln und Ausnahmefälle zu nennen, würde den Rahmen dieses Buches sprengen, schließlich wollen Sie keine mathematische Abhandlung lesen, sondern mehr zur Grafikprogrammierung erfahren.

Für unsere speziellen Fälle werden jedoch die Addition und besonders die Multiplikation von rechteckigen Matrizen wichtig, so daß beide Operationen an einem Beispiel vorgeführt werden sollen.

Betrachten wir die beiden Matrizen M1 und M2:

```
        [-1 0]        [0  1]
M1=             M2=
        [1  0]        [1  1]
```

Bei der Addition (Subtraktion) addiert (subtrahiert) man nun die Elemente mit demselben Zeilen- wie auch Spaltenindex. In unserem Falle lautet das Ergebnis daher:

```
[-1  1]

[2   1]
```

Die Multiplikation mutet auf den ersten Blick etwas seltsam an, erweist sich nach einiger Übung aber auch nicht mehr als Problem: Man multipliziert die Elemente einer Zeile der ersten Matrix mit den Elementen einer Spalte der zweiten Matrix. Die resultierenden Produkte werden anschließend addiert:

```
[-1 * 0] + [0 * 1]     [-1 *1] + [0 *1]

[1  * 1] + [0 * 1]     [1 *1]  + [0 *1]

              0  -1

             -1   1
```

Allgemein ausgedrückt, ergibt sich somit folgende Rechenvorschrift:

```
[A   B]       [a   b]       A*a+B*c    A*b+B*d
          *             =
[C   D]       [c   d]       C*a+D*c    C*b+D*d
```

Eine Transformation wie die eben durchgeführte Parallelverschiebung läßt sich nun recht gut als Produkt zweier Matrizen darstellen, allerdings handelt es sich dann um eine Multiplikation zwischen einer 1-mal-2-Matrix und einer 2-mal-2-Matrix:

```
              [a   b]
x     y   *             =   a*x+c*y    b*x+d*y
              [c   d]
```

X und y sind dabei die Koordinaten eines jeden Originalpunktes, die Summen auf der rechten Seite entsprechen xneu und yneu.

Welcher Art die durchgeführte Transformation ist, wird durch die Werte der zweiten Matrix festgelegt; die Möglichkeiten zeigt das folgende Programm.

```
5       rem matrixdemo
6       rem --------------------
10      clearw 2:fullw 2
18      ' neues Bildschirm-Koordinatensystem
19      ' mit vier Quadranten:
20      xmin = -320 : xmax = 320
30      ymin = -200 : ymax = 200
50      start:
60      clearw 2
70      gosub achsenkreuz
80      gosub bild.lesen
```

```
90    gosub zeichnen
99    '
100   menue:
110   gotoxy 0,0:print "MATRIXDEMO"
120   input "a ";a : input "b ";b
130   input "c ";c : input "d ";d
140   clearw 2
150   gotoxy 0,0:print "Matrix: "
160   print a,b : print c,d
170   gosub transform
180   input "Originalbild";e$:if left$(e$,1)="j" then goto start
190   gotoxy 0,0:for z=1 to 4 :print string$(20," "):next:goto menue
200   bild.lesen:
210   restore : read anz
220   for punkt=1 to anz
230   read x,y : x(punkt) = x : y(punkt) = y
240   next punkt
250   return
299   '
300   achsenkreuz:
310   xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
320   linef xv,yv,xn,yn  : rem x-Achse
330   xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
340   linef xv,yv,xn,yn  : rem y-Achse
350   return
399   '
600   zeichnen:
610   for punkt=2 to anz
620   xn = x(punkt):yn=y(punkt):xv= x(punkt-1):yv=y(punkt-1)
630   gosub transline: linef xv,yv,xn,yn
640   next punkt
650   return
699   '
700   transform:
710   for punkt=1 to anz
720   x(punkt) = a * x(punkt) + c * y(punkt)
730   y(punkt) = b * x(punkt) + d * y(punkt)
740   next punkt
750   gosub achsenkreuz
760   gosub zeichnen
```

```
770   return
9999  '
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
10050 transline:
10060 xv = 640 * (xv - xmin) / (xmax - xmin)
10070 yv = 400 * (ymax - yv) / (ymax - ymin)
10080 xn = 640 * (xn - xmin) / (xmax - xmin)
10090 yn = 400 * (ymax - yn) / (ymax - ymin)
10100 return
20000 data 8,0,0,0,80,50,130,100,80,100,0,0,0,0,80,,100,80
```

Nachdem Sie das Programm Matrixdemo korrekt eingegeben und gestartet haben, wird zunächst ein Achsenkreuz gezeichnet. Dann werden die Daten unseres Hausumrisses in den indizierten Variablen x und y abgespeichert (Zeilen 200 bis 250) und es wird das Originalbild gezeichnet.

Um die Eingaben des Benutzers entgegennehmen zu können, wird innerhalb der Routine Menü der Cursor an der Position 0,0 plaziert; anschließend werden Sie zur Eingabe von a, b, c und d aufgefordert. Im folgenden wird das Schema der Matrix ausgedruckt bevor die neue Abbildung nach Transformierung der Koordinaten gezeichnet wird.

Probieren Sie zunächst die unterschiedlichsten Eingaben aus und versuchen Sie, Gesetzmäßigkeiten festzustellen. Allerdings ist es dabei zweckmäßig, den Wertebereich auf Zahlen zwischen -2 und 2 zu begrenzen, außerdem sollten Sie systematisch vorgehen und nicht gleich alle vier Parameter mit Zufallszahlen belegen.

### 3.3.1 Maßstabsänderungen

Wir hatten bereits festgestellt, daß die Multiplikation sämtlicher x-Werte mit einem konstanten Faktor das Bild in Richtung der x-Achse vergrößert (gleiches gilt natürlich auch für alle y(i) auf der y-Achse). Entsprechend muß die Multiplikation mit einer Zahl kleiner eins zu einer Verkleinerung des Bildes führen.

Betrachten wir nun unsere Matrix unter diesem Aspekt, so stellen wir fest, daß diese Faktoren wohl mit a und d identisch sind (setzen Sie für b und c Null ein):

```
          [a   0]
x y  *              =  a*x+0*y , 0*x+d*y = a*x , b*y
          [0   d]
```

Mit den Werten für a und d legen wir also die Größe des Bildnisses fest, wobei a für eine Vergrößerung/Verkleinerung in x-Richtung, und d für eine Maßstabsänderung entlang der y-Achse stehen. Dabei bedeuten Werte zwischen 0 und 1 eine Verkleinerung und Werte zwischen 1 und unendlich eine Vergrößerung.

Die folgende Abbildung zeigt somit ein Haus von doppelter Breite und nur einem drittel der Originalhöhe.

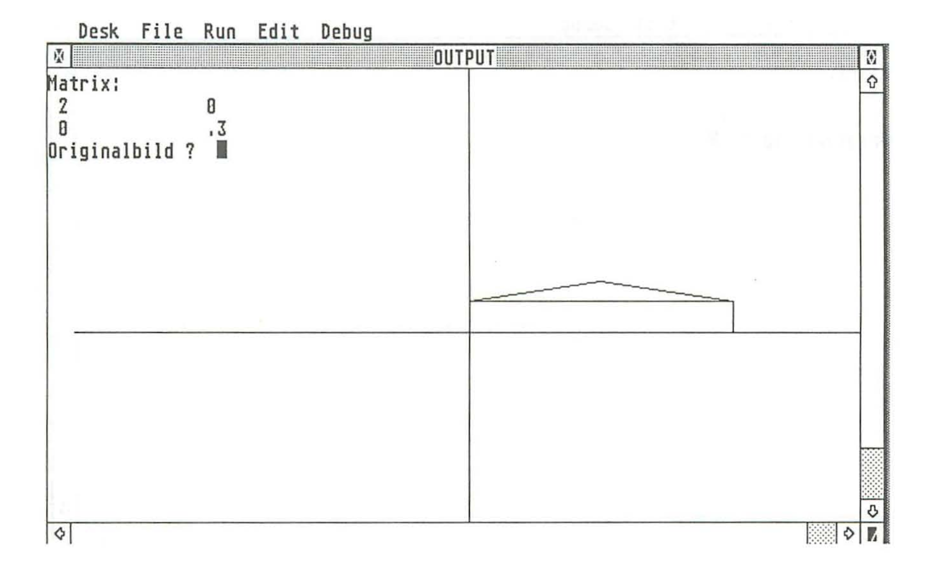
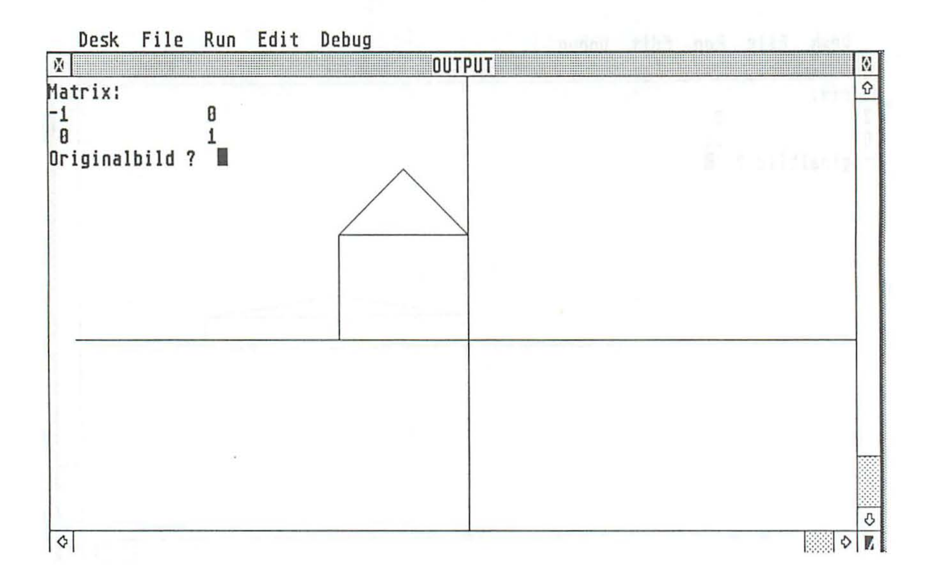



Abb. 29: Transformation mit M = 2 0 0 0.3

### 3.3.3.2 Spiegelungen

Demnach läßt sich auch eine 'neutrale Matrix' festlegen, denn wenn beide Faktoren gleich eins sind, dann wird die Abbildung identisch mit dem Originalbild sein.

Je nach Vorzeichen einer solchen Matrizenmultiplikation wird das Ergebnis dann eine Spiegelung an der y-Achse, der x-Achse oder durch den Koordinatenursprung sein (vgl. dazu Abb. 28 und Abb. 29).



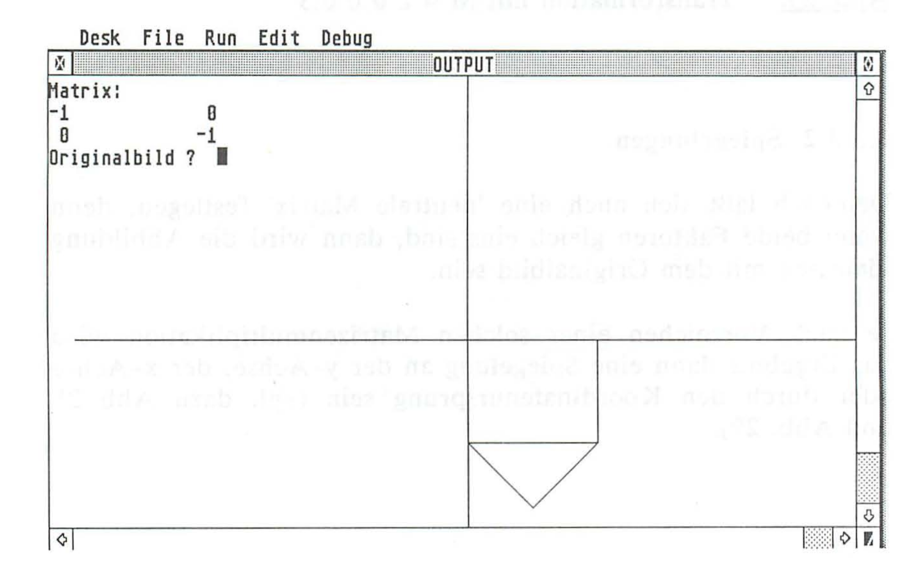




Abb. 30: Auswirkungen von -1 0 0 1 und -1 0 0 -1

### 3.3.3.3 Scherung

Interessant wird es nun, wenn Sie sich daran machen, die verschiedensten Werte für b und c einzusetzen. Damit können Sie die abzubildenden Objekte in eine beliebige Schräglage bringen.

Auch hier soll wieder das Schema zur Berechnung der einzelnen Koordinaten angegeben werden:

$$x \quad y \quad * \quad \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} \quad = \quad x,\ b * x + y$$

Da a und b beide gleich eins sind, wird keine Maßstabsänderung vorgenommen, wohl aber wird durch das Matrixelement b die y-Achse mit wachsendem x zunehmend verschoben.

Entsprechendes gilt für die x-Achse, wenn c ungleich null gewählt wird.

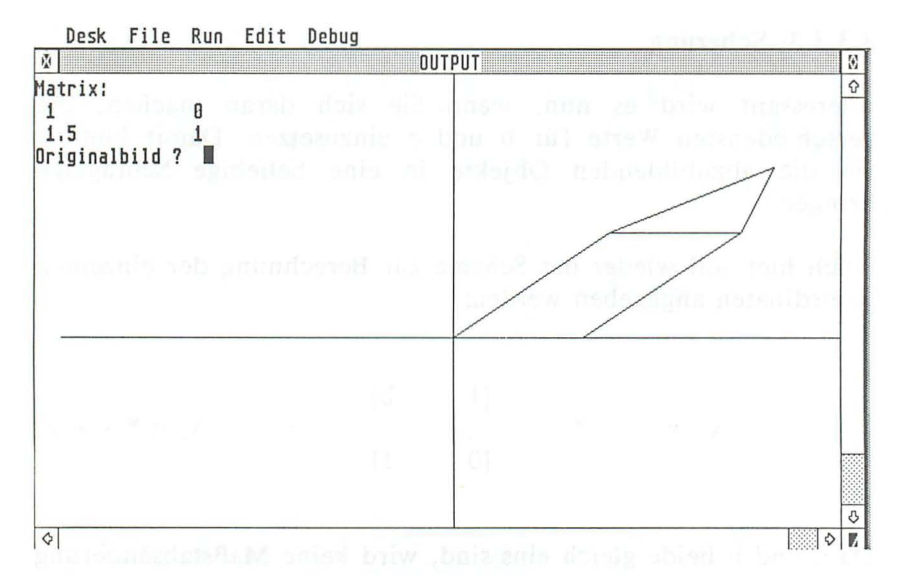



Abb. 31: Auswirkungen der Matrix 1 0 1.5 1

### 3.3.3.4 Rotation

Was uns an grundlegenden Techniken zur Veränderung einer Darstellung noch fehlt, ist eine Vorschrift für die Rotation des Bildnisses um eine Achse beziehungsweise um den Ursprung des Koordinatensystemes. Tatsächlich haben wir bereits eine entsprechende Transformation vorgenommen, denn unsere zweite Methode, um einen Kreis auf den Bildschirm zu bringen, tut nichts anderes, als einen (einzigen) Punkt in einer konstanten Entfernung (Radius) um den Koordinatenursprung (Mittelpunkt) zu drehen.

Die Matrix, die eine solche Rotation repräsentiert, können wir der Trigonometrie entnehmen.

Unter Trigonometrie, oder Drehwinkelmessung, versteht man das Messen und Berechnen von Dreiecken. Handelt es sich dabei um rechtwinklige Dreiecke, so ist die Berechnung aller Werte aus zwei Teilangaben besonders einfach. Denn zum einen

müssen die Winkel an der Hypotenuse zusammen 90 Grad ergeben, zum anderen gilt der pythagoreische Lehrsatz.

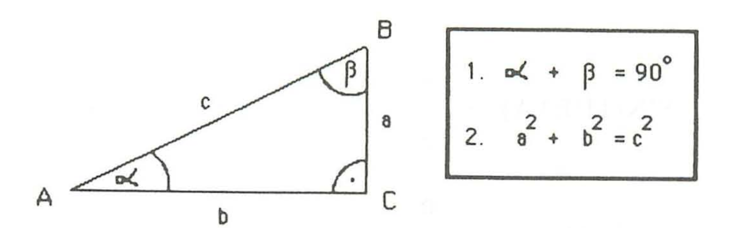



Abb. 32: Satz des Pythagoras

Um nun die Sätze der Trigonometrie anwenden zu können, kann jede durch gerade Linien begrenzte Figur in Dreiecke zerlegt werden, was insbesondere auch für jeden Punkt innerhalb eines kartesischen Koordinatensystemes gilt:

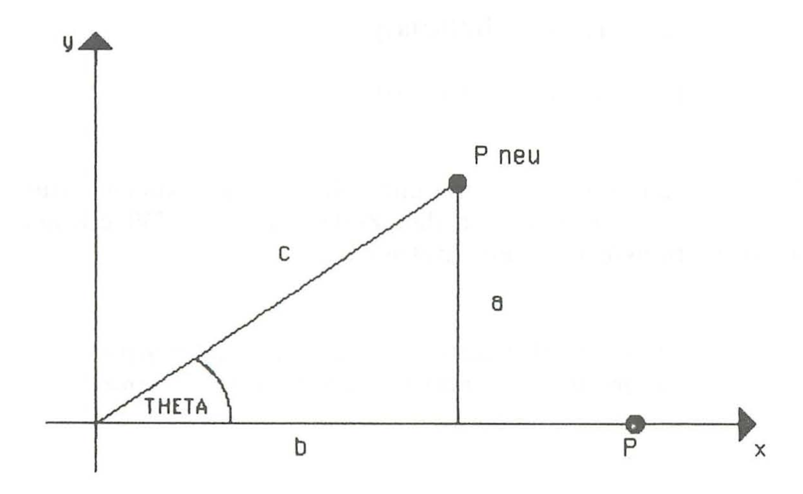



Abb. 33: Winkel im Koordinatensystem

Die trigonometrischen Funktionen SIN und COS bilden nun die Verhältnisse der Seiten in einem rechtwinkligen Dreieck.

So gilt:

$$\mathrm{SIN(THETA)} = \frac{a}{c}$$

$$\mathrm{COS(THETA)} = \frac{b}{c}$$

Theta ist der gewünschte Drehwinkel, a entspricht einem Teilstück der y-Achse und b einem Teilstück der x-Achse.

Unsere Matrix lautet daher:

[cos(theta) sin(theta)]

[-sin(theta) cos(theta)]

Falls Sie nun mit der Drehung ein wenig experimentieren möchten, so reicht es aus, in den Zeilen 720 und 730 die neuen Transformationsformeln einzusetzen:

```
720 xneu(punkt)=cos(a)*x(punkt) + (-sin(c))* y(punkt)
730 yneu(punkt)=sin(b) * x(punkt) + cos(d) * y(punkt)
```

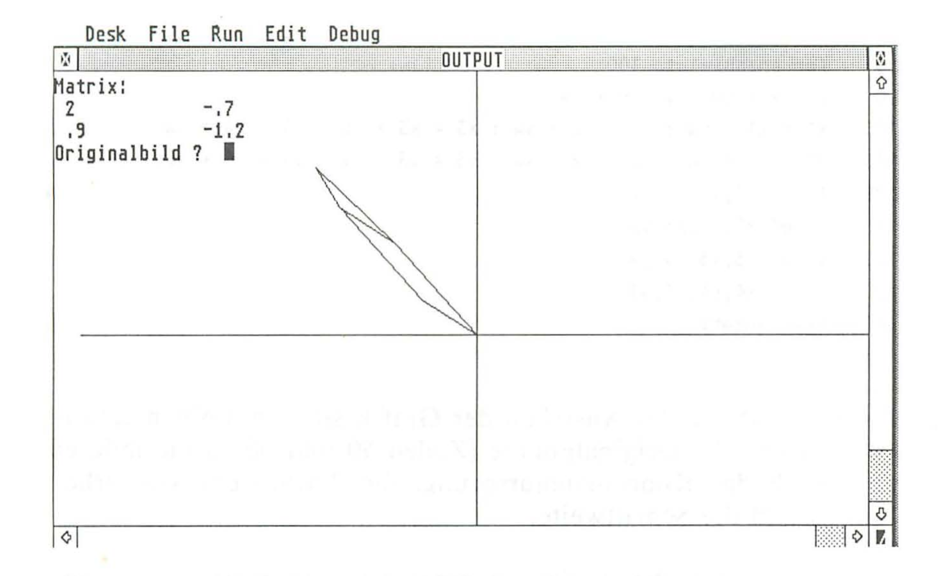



Abb. 34: Hardcopy zu Matrixdemo

### 3.3.3.5 Anwendungen

Shapes, Matrizen und Transformationsgleichungen lassen sich nicht nur in Programmen zum darstellenden Zeichnen von Objekten (CAD) finden, sondern Shapes und Koordinatentransformationen sind meistens auch das Kernstück eindrucksvoller Grafikprogramme.

Die Vorgehensweise sieht dabei so aus, daß ein einfaches Shape, ein Dreieck oder Viereck, manchmal aber auch ein Vieleck, um (s)einen Mittelpunkt gedreht wird:

```
10    rem computergrafik 3
20    rem viereck; gedreht und vergroessert
30    rem ------------------------------
40    clearw 2:fullw 2
50    x1 = 100 : x2 = 400 : x3 = 400 : x4 = 100
60    y1 = 300 : y2 = 300 : y3 = 100 : y4 = 100
```

```
70    sw = 5
80    for anzahl=1 to 100
90    x = x + sw : y = y + sw
100   x1 = x1 - sw : x2 = x2 - sw : x3 = x3 + sw : x4 = x4 + sw
110   y1 = y1 + sw : y2 = y2 - sw : y3 = y3 - sw : y4 = y4 + sw
120   linef x1,y1,x2,y2
130   linef x2,y2,x3,y3
140   linef x3,y3,x4,y4
150   linef x4,y4,x1,y1
160   next anzahl
```

Bestimmend für das Aussehen der Grafik sind zum einen selbstverständlich die Originalpunkte (Zeilen 50 und 60), zum anderen aber auch der Koordinatenursprung, die Anzahl der Wiederholungen und die Schrittweite.

So kann bereits die Änderung eines einzigen Wertes zu einem ganz anderen Bild führen!

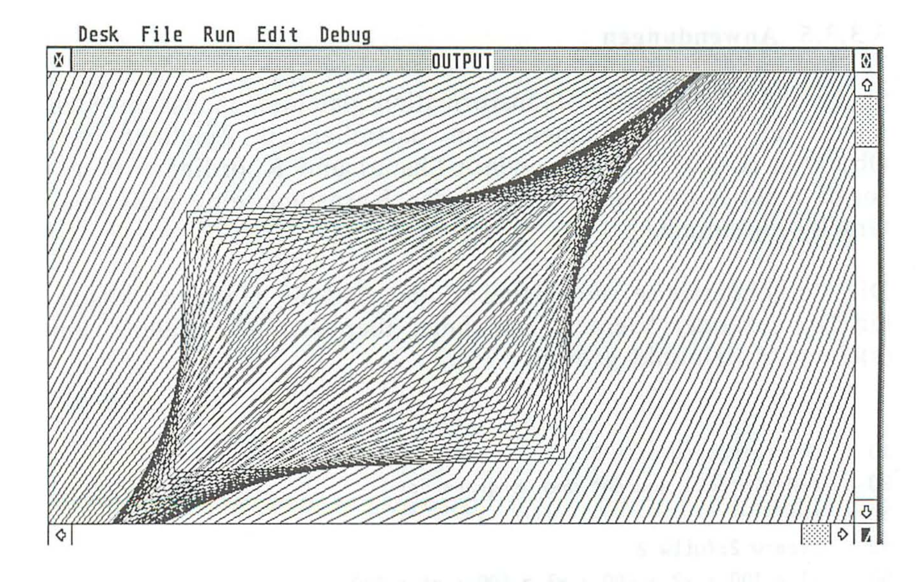



Abb. 35: Hardcopy zu GRAFIK3.BAS

Nun erweist sich dieses Programm noch als recht unflexibel und gewöhnungsbedürftig. Werden die Daten doch nicht innerhalb eines Arrays gespeichert und lassen sich auch unsere Transformationsformeln nicht sogleich entdecken.

Ganz anders sieht es jedoch bei folgendem Programm aus:

```
10    rem computergrafik 4
20    rem vielecke; gedreht und vergroessert
30    rem ---------------------------------
35    rem Koordinatenursprung in Bildmitte
40    xmin=-320:xmax=320:ymin=-200:ymax=200
45    '
49    ' Daten einlesen
50    clearw 2 : fullw 2 : read anz
60    for p=1 to anz
70    read xo(p),yo(p)
80    next p
89    '
90    rem Koordinatentransformation
100   rem wie oft und um wieviel drehen
110   for theta=1 to 180 step 3
119   '
120   rem alle Eckpunkte umrechnen
130   for p=1 to anz
140   x = xo(p) * cos(theta) - yo(p) * sin(theta)
150   y = xo(p) * sin(theta) + yo(p) * cos(theta)
160   gosub transpoint : x(p)=x : y(p) = y : next p
170   gosub zeichnen
180   next theta
190   end
199   '
200   zeichnen:
210   linef x(1),y(1),x(2),y(2)
220   linef x(2),y(2),x(3),y(3)
230   linef x(3),y(3),x(1),y(1)
240   return
250   '
```

```
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
20000 rem hier folgen die Bilddaten
20010 rem zuerst die Anzahl der Eckpunkte
20020 data 4,-100,-100,100,-100,100,100,-100,100
```

Die Daten des Shapes, das wir rotieren und skalieren wollen, stehen wie gewohnt am Ende des Programmes; damit die Werte auch richtig eingelesen werden, steht zu Beginn wieder die Anzahl der Punkte. Gelesen und den Variablen xo und yo zugewiesen werden die Eckpunkte innerhalb der Schleife von 60 bis 80. Der Drehwinkel wird in Zeile 110 festgelegt, und in den Zeilen 140 und 150 erkennen wir auch die Transformationsgleichungen wieder.

Sie sehen, daß hier eine Drehung durchgeführt wird, was allein schon zu ansprechenden Grafiken führt. Sollten Sie sich jedoch an den Bildern sattgesehen haben, können Sie das Shape darüber hinaus auch noch von Schritt zu Schritt vergrößern oder verkleinern lassen.

Einen weiteren Ansatzpunkt bietet unsere Transpoint-Routine. Schließlich müssen Sie den Koordinatenursprung nicht in der Schirmmitte liegen lassen.

Bei all diesen Änderungen sollten Sie jedoch nicht vergessen, daß auch hier wieder die Daten das Bild machen, probieren Sie beispielsweise auch:

```
20020 data 4,-100,-100,100,-100,100,100,-100,100
```

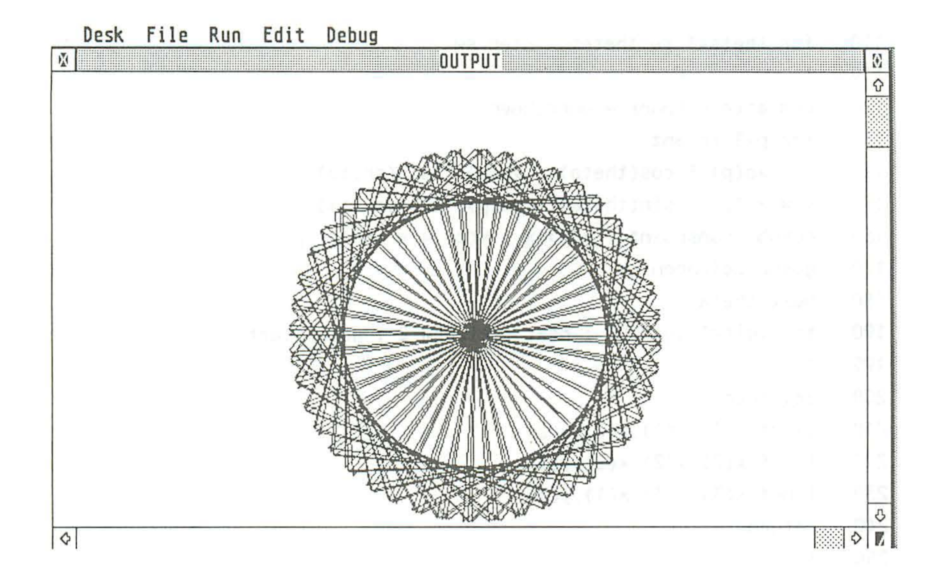



Abb. 36: Ein Beispielausdruck zu GRAFIK4.BAS

Die vielfältigen Möglichkeiten, die allein bei der Drehung von Shapes auftreten können, zeigt folgendes Programm in einer unablässigen Show:

```
10    rem spirograf
20    rem vielecke um Koordinatenursprung gedreht
30    rem ---------------------------------------
35    rem Koordinatenursprung in Bildmitte
40    xmin=-320:xmax=320:ymin=-200:ymax=200:fullw 2
50    dim xo(15),yo(15),x(15),y(15)
59    ' wie viele Eckepunkte?
60    start:
70    clearw 2 : anz = rnd(1) * 15
79    ' Ausgangsbild erzeugen:
80    for p=1 to anz : xo(p) = rnd(1) * 320 - 160: yo(p) = rnd(1) * 200-
100:next p
90    rem Koordinatentransformation
99    rem wie oft und um wieviel drehen
100   thetamax = rnd(1) * 320 : sw = rnd(1) * 10
```

```
110   for theta=1 to thetamax step sw
119   '
120   rem alle Eckpunkte umrechnen
130   for p=1 to anz
140   x = xo(p) * cos(theta) - yo(p) * sin(theta)
150   y = xo(p) * sin(theta) + yo(p) * cos(theta)
160   gosub transpoint : x(p)=x : y(p) = y : next p
170   gosub zeichnen
180   next theta
190   for zeit=1 to 8000 : next : clearw 2 : goto start
199   '
200   zeichnen:
210   linef x(1),y(1),x(2),y(2)
220   linef x(2),y(2),x(3),y(3)
230   linef x(3),y(3),x(1),y(1)
240   return
250   '
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
```

## 3.4 Funktionen

Die wohl häufigste Anwendung im Bereich der zweidimensionalen Computergrafik ist wohl das Plotten von Funktionen. Darunter versteht man die grafische Darstellung der Beziehung zwischen jeweils zwei Werten zueinander, wobei die Art der Beziehung meistens durch eine mathematische Formel angegeben werden kann.

So wurde denn auch der Begriff der Funktion bereits zu Beginn des 18. Jahrhunderts definiert. Doch erwies sich die damals formulierte Beschreibung einer Funktion als 'veränderliche Größe, die von einer anderen veränderlichen Größe abhängt' als nicht exakt genug für die späteren Ansprüche der Mathematik. Denn charakteristisch für das Wesen der Funktion sind nicht die

unterschiedlichen Größen (Zahlenwerte), sondern vielmehr die Zuordnung dieser Größen zueinander.

So ist es auch zu erklären, daß nicht nur Zahlen, sondern auch eine Reihe von Gegenständen per Funktion zugeordnet werden können. Natürlich wird ein Mathematiker niemals von einer Reihe, sondern von einer Menge sprechen, und er wird uns erklären:

"Eine Funktion ordnet jedem Element einer Menge ein bestimmtes Element einer anderen Menge zu."

Als Beispiel könnte hier die Menge der Autos auf der einen, und die Menge der Fahrer auf der anderen Seite betrachtet werden, denn zu jedem Auto läßt sich ein Fahrer nennen. Leider ist diese recht schöne Beziehung noch keine Funktion, denn jedem Auto kann nicht 'ein bestimmter' Fahrer zugewiesen werden. Besser ist es da schon, wenn man die Autos in Beziehung zu den augenblicklich eingetragenen Haltern setzt, denn dann kann in Zusammenhang mit jedem Auto auch nur eine einzige Person genannt werden.

Nennen wir die Autos nun x und die Kfz-Halter y, dann läßt sich eine Definition der Funktionen folgendermaßen formulieren:

Eine Funktion ist eine eindeutige Abbildung zweier Mengen, bzw. einer gewissen Menge von geordneten Paaren (x,y), für die gilt, daß es zu jedem x genau ein y gibt.

Symbolisch wird der Zusammenhang als

$$y = f(x)$$

dargestellt, wobei x den Elementen der ersten Menge entspricht, eben allen den Elementen, für die die Funktion definiert ist, und y stellvertretend für die errechneten Werte steht. Analog dazu spricht man auch von Definitions- und Wertebereich.

### 3.4.1 Das Plotten von 2-D-Funktionen

Jedem Schüler werden heute zumindest drei Methoden zur Darstellung von Funktionen nahebebracht.

Schnell wieder vergessen wird der Funktionsgraph, bei dem der Definitionsbereich und der Wertebereich in Form von Kreisen oder Ovalen dargestellt und die Zuordnungen in Form von Pfeilen veranschaulicht werden. Eine Funktion erkennt man bei dieser Darstellungsform dann daran, daß von jedem Element der Definitionsmenge nur ein einziger Pfeil ausgeht; dennoch können mehrere Pfeilspitzen auf ein Element der Ergebnismenge weisen.

Nachdem solch ein Graph dann erstellt war, ließ sich aus ihm leicht eine Wertetabelle erstellen: Die einzelnen Elemente des Definitionsbereiches wurden neben- oder untereinander notiert und der per Pfeil zugeordnete Wert dann dazugeschrieben.
In vielen Fällen ließ sich dann jedem Zahlenpaar dieser Wertetafel ein Punkt P in der Ebene zuordnen, wodurch nach und nach ein Bild der Funktion entstand.

Üblich in den meisten Fällen war dabei die Verwendung des kartesischen Koordinatensystems, wobei die x-Werte auf der Waagerechten und die durch die Funktionsgleichung festgelegten Werte auf der senkrechten Achse abgetragen wurden. Abhängig von der Beschaffenheit des Definitionsbereiches und der Funktionsgleichung erhielt man dann eine Reihe von Punkten, Kurvenstücken oder auch eine ausgezeichnete Funktionskurve.

### 3.4.2 Ein Funktionenplotter

Alles, was wir im folgenden tun müssen, ist, diese Technik in ein Programm umzusetzen und dabei die Gegebenheiten des Computers zu berücksichtigen.

Der erste Schritt wäre demnach die Erstellung einer Wertetafel:

```
10 FULLW 2 : CLEARW 2
200 FOR x=1 TO 10
210 y = x * x
220 PRINT x;y
230 NEXT x
```

In Zeile 200 legen wir unseren Definitionsbereich fest, in diesem Beispiel von 1 bis 10, in Zeile 210 steht die Funktion und wird f(x) berechnet, und Zeile 220 ist dafür verantwortlich, daß sämtliche Werte in Form einer Tabelle untereinander ausgegeben werden.

Auch eine einfache Skizzierung der Funktion ist bereits möglich; ersetzen Sie Zeile 220 durch die Anweisung PRINT TAB(y)"*". Nach dem Start des kurzen Programmes werden Sie den berechneten Ast der Normalparabel ohne Schwierigkeiten erkennen.

Ähnlich wären Sie auch bei einer Abbildung der Funktion auf Papier vorgegangen. Sie hätten nämlich ein rechtwinkliges Koordinatensystem gewählt, bei dem die x-Werte von links nach rechts an- und die y-Werte von unten nach oben aufsteigen. Sie hätten zunächst die Koordinatenachsen und daran anschließend die berechneten Punkte gezeichnet. Schließlich hätten Sie durch alle so abgebildeten Punkte eine geschwungene Linie gezogen.

Wie wir bereits festgestellt haben, entspricht das Koordinatensystem des ST nicht gerade unserem, bevor ein Punkt auf dem Schirm abgebildet wird, werden wir erst wieder unsere Transformationsgleichung aufrufen müssen. Ein Koordinatenkreuz hatten wir ebenfalls schon gezeichnet - also sollte es doch

reichen, das Programm um die früheren Routine zu ergänzen und die PRINT-Anweisung in 220 durch eine Anweisung zum Setzen eines einzelnen Punktes zu ersetzen:

```
20    rem fplot1
30    rem --------------------
40    clearw 2:fullw 2
50    xmin = -320 : xmax = 320
60    ymin = -200 : ymax = 200
70    gosub achsenkreuz
200   for x1=-10 to 10
210   x = x1 : y = x1 * x1
220   gosub transpoint
230   linef x,y,x,y
240   next x1
290   end
300   achsenkreuz:
310   xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
320   linef xv,yv,xn,yn  : rem x-Achse
330   xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
340   linef xv,yv,xn,yn  : rem y-Achse
350   return
9999  '
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
10050 transline:
10060 xv = 640 * (xv - xmin) / (xmax - xmin)
10070 yv = 400 * (ymax - yv) / (ymax - ymin)
10080 xn = 640 * (xn - xmin) / (xmax - xmin)
10090 yn = 400 * (ymax - yn) / (ymax - ymin)
10100 return
```

Tatsächlich erscheint die Parabel korrekt auf dem Schirm, nur die Größe des Abbildes entspricht nicht so ganz unseren Erwartungen. Doch ist auch das für uns inzwischen kein Problem mehr, denn, daß sich Skalierungen durch Multiplikation

mit einem konstanten Faktor bewältigen lassen, haben wir inzwischen gelernt.

Lassen wir die Kurve doch vergrößert zeichnen und legen wir geeignete Maßstabsfaktoren in Zeile 80 fest:

```
80 mfx=20 : mfy = 5
210 x = x1 * mfx : y = x1 * x1 * mfy
```

Dem erzeugten Bild nach läßt sich annehmen, daß die Lage der gesetzten Punkte annähernd richtig ist, doch muß man eine Funktion schon recht gut kennen, um solche Schlüsse aus einem Minimum an Punkten ziehen zu können. Was noch fehlt, sind einfach mehr Punkte - um nicht zu sagen möglichst viele.

Ergänzen wir Zeile 200 daher um STEP .1, und schon:

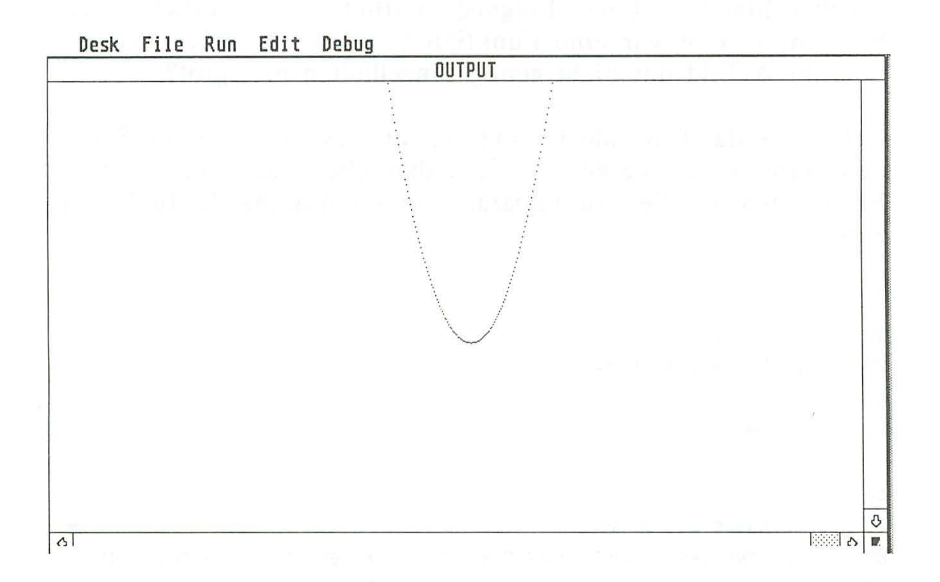



Abb. 37: Die Normalparabel - schnell geplottet

Um nun einen geschlossenen Linienzug zu erhalten, stehen uns zwei Wege offen: Wir könnten die zwischen den bereits gezeichneten Punkten liegenden Punkte berechnen, oder wir lassen den Computer unsere eigene Arbeitsweise übernehmen und ihn eine Verbindungslinie von Punkt zu Punkt ziehen.

Bei Anwendung des letzten Verfahrens ist unser Programm zwar schnell, aber leider ungenau, denn der Computer kann die Krümmung der Kurve nicht abschätzen wie wir es tun, sondern er zeichnet immer eine Gerade. Und das andere Lösungsverfahren ist zwar exakt, doch benötigt das Programm bei ausreichend kleinen Schrittweiten aufgrund der enormen Rechenarbeit eben auch entsprechend viel Zeit. Überzeugen Sie sich davon, indem Sie die Zeilen 200 und 210 austauschen:

```
200 FOR x=-10 TO 10 STEP 0.01
210 y=SIN(x)
```

Doch wie wäre es mit einer Kombination beider Vorschläge und darüber hinaus mit der Eingabe sämtlicher Kontrollwerte nach Wunsch, so daß wir eine Funktion beliebig vergrößern können, falls ihr Abbild uns nicht genug Einzelheiten preisgibt?

Geben Sie das folgende Listing ein, und experimentieren Sie mit dem Programm. Vergessen Sie dabei aber auch nicht, andere Funktionen als die Normalparabel in Programmzeile 10000 einzusetzen.

```
10      rem fplot2
20      rem mit Autoscaling
30      rem ---------------
40      clearw 2
50      fullw 2
60      '
70      ' Eingabe plotdaten
80      input"Wertebereich von x= ";x: xmin = x: gosub funktion : ymin = x
90      input" bis x= ";x: xmax = x: gosub funktion : ymax = x
```

```
100   input"Genauigkeit (0.1 - 0.01) ";genauigkeit
110   input"Vergroesserungsfaktor (1) ";mf
120   clearw 2:gosub achsenkreuz
130   x1=xmin:gosub funktion:y=y*mf:x=x1*mf:gosub transpoint:xalt=x:yalt
=y
140   for x1=xmin to xmax step genauigkeit
150   gosub funktion
160   y = y * mf : x = x1 * mf
170   gosub transpoint
180   linef xalt,yalt,x,y : xalt = x : yalt = y
190   next x1 : goto skalierung
200   '
210   achsenkreuz:
220   xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
230   linef xv,yv,xn,yn  : rem x-Achse
240   xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
250   linef xv,yv,xn,yn  : rem y-Achse
260   return
270   '
280   skalierung:
290   xmin = -320 : xmax = 320 :ymin = -200 : ymax = 200
300   for xa=-320 to 320 step 64
310   xv = xa : yv = 5 : xn = xa : yn = -5
320   gosub transline
330   linef xv,yv,xn,yn
340   next xa: end
350   '
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
10040 transline:
10050 xv = 640 * (xv - xmin) / (xmax - xmin)
10060 yv = 400 * (ymax - yv) / (ymax - ymin)
10070 xn = 640 * (xn - xmin) / (xmax - xmin)
10080 yn = 400 * (ymax - yn) / (ymax - ymin)
10090 return
20000 funktion:
20010 y = sin(x1)
20020 return
```

Eigentlich funktioniert doch nun schon alles recht ordentlich. Nur eine Beschriftung der Achsen fehlt noch, um das Bild aussagekräftiger zu machen. Dazu gibt es wiederum mehrer Techniken, von denen eine hier vorgestellt werden soll.

So kann zunächst festgelegt werden, mit wie vielen Teilstrichen jede Achse markiert werden sollen. In unserem Fall sollen es jeweils zehn sein, was bedeutet, daß sich auf der x-Achse alle 64 x-Einheiten eine Markierung befinden muß. Für die y-Achse beträgt der Abstand von Markierung zu Markierung demgemäß 40 Einheiten.

Nach dieser Vorüberlegung kann eine Schleife aufgebaut werden, welche die Skalierung vornimmt. Dabei wird der Zahlenwert einer jeden Markierung der Quotient aus der Anzahl der Zeicheneinheiten durch den Maßstabsfaktor sein. Wurde der Wert so errechet, kann die Beschriftung wie gehabt mittels des angenommenen Grafik-Cursors vorgenommen werden.

Außerdem muß dabei ein anderer Schreibmodus eingeschaltet werden, den wir im nächsten Kapitel noch genauer kennenlernen werden. Es handelt sich dabei um eine Betriebsart, die eine 'ODER-Verknüpfung' zwischen vorhandenen und neu zu zeichnenden Bildteilen ausführt, um zu verhindern, daß Teile des Funktionsbildes gelöscht werden.

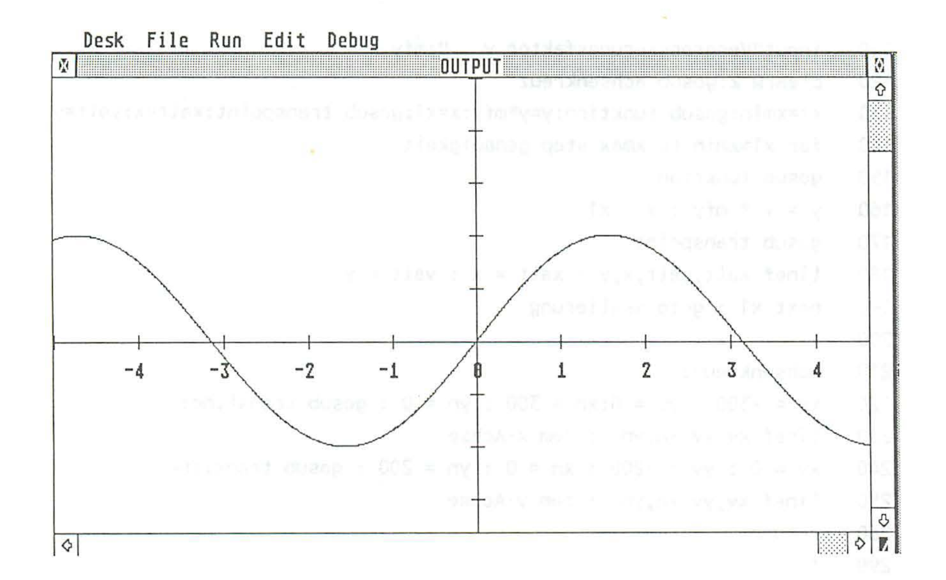



Abb. 38: Ein Beispielausdruck des Plotters

Ein letztes Problem kann eine bei der Berechnung der Funktionswerte auftretende Division durch null sein. Dieser Fehler würde normalerweise zum Programmabbruch führen. Deshalb wird innerhalb von Plotprogrammen entweder zu jedem Wert des Definitionsbereiches ein konstanter Faktor (vielleicht 0.00001) addiert, oder das Programm wird angewiesen, bei Auftreten eines solchen Fehlers sogleich mit dem nächsten Wert weiterzuarbeiten.

```
10     rem plotter
20     rem mit Autoscaling
30     rem ----------------
40     clearw 2
50     fullw 2
60     on error goto 10100 : rem im Falle eines Falles ...
70     ' Eingabe plotdaten
80     input"Wertebereich von x= ";x: xmin = x: gosub funktion : ymin = x
90     input" bis x= ";x: xmax = x: gosub funktion : ymax = x
100    input"Genauigkeit (0.1 - 0.01) ";genauigkeit
```

```
110   input"Vergroesserungsfaktor y   ";mfy
120   clearw 2:gosub achsenkreuz
130   x1=xmin:gosub funktion:y=y*mfy:x=x1:gosub transpoint:xalt=x:yalt=y
140   for x1=xmin to xmax step genauigkeit
150   gosub funktion
160   y = y * mfy : x = x1
170   gosub transpoint
180   linef xalt,yalt,x,y : xalt = x : yalt = y
190   next x1 : goto skalierung
200   '
210   achsenkreuz:
220   xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
230   linef xv,yv,xn,yn   : rem x-Achse
240   xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
250   linef xv,yv,xn,yn   : rem y-Achse
260   return
290   '
300   skalierung:
310   gosub transparent.modus
320   if xmin<0 then xmin = -xmin:diff = xmin + xmax
330   strich=0 : for xa=-xmin to xmax step diff/10
340   strich=strich+1
350   label$(strich)=str$(xa)
360   next xa
370   label$(1)="":label$(11)="": rem am Rand keine Beschriftung
380   xmin = -320 : xmax = 320 :ymin = -200 : ymax = 200
390   strich = 0 : for xa=xmin to xmax step 64 : rem 64 = 10 Teilstriche
400   strich = strich +1
410   xv = xa : yv = 5 : xn = xa : yn = -5
420   gosub transline
430   linef xv,yv,xn,yn
440   text$=left$(label$(strich),6)
450   y = yn + 60 : x = xn - 10
460   gosub v.gtext
470   next xa
480   for ya = 40 to 200 step 40
490   yv= ya : yn =yv: xv = -5 : xn = 5
500   gosub transline
510   linef xv,yv,xn,yn : linef xv,400-yv,xn,400-yn
520   next ya
```

```
530   gosub replace.modus
540   end
5990  '
6000  v.gtext:
6010  for digit=1 to len(text$)
6020  poke intin + (digit - 1) * 2,asc(mid$(text$,digit,1))
6030  next digit
6040  poke intin + (digit - 1) * 2,0
6050  poke contrl,8
6060  poke contrl+2,1
6070  poke contrl+6,len(text$)+1
6080  poke ptsin,x
6090  poke ptsin+2,y
6100  vdisys
6110  return
6980  ' Schreibmodus setzen
6990  '
7000  transparent.modus:
7010  poke intin,2
7020  poke contrl,32
7030  poke contrl+2,0
7040  poke contrl+6,1
7050  vdisys 32
7060  return
7070  '
7080  replace.modus:
7090  poke intin,1
7100  poke contrl,32
7110  poke contrl+2,0
7120  poke contrl+6,1
7130  vdisys 32
7140  return
9990  '
10000 transpoint:
10010 x = 640 * (x - xmin) / (xmax - xmin)
10020 y = 400 * (ymax - y) / (ymax - ymin)
10030 return
10040 transline:
10050 xv = 640 * (xv - xmin) / (xmax - xmin)
10060 yv = 400 * (ymax - yv) / (ymax - ymin)
```

```
10070 xn = 640 * (xn - xmin) / (xmax - xmin)
10080 yn = 400 * (ymax - yn) / (ymax - ymin)
10090 return
10100 resume next      : rem ... einfach weitermachen!
19990 ' Funktion als f(x1) in Zeile 20010 einsetzen.
20000 funktion:
20010 y = sin(x1)
20020 return
```

# 4. Kapitel

# 3-D-Grafik

## 4.1 Die dritte Dimension

Zur dreidimensionalen Darstellung von Objekten haben sich im Laufe der Jahre eine Reihe von Verfahren entwickelt, die sich zum Teil auch auf Computern einsetzen lassen. Doch handelt es sich meist nur um die Illusion eines dreidimensionalen Gebildes denn wie sollte eine zweidimensionale Zeichenfläche in die Tiefe gehen können? Und so sieht man in den meisten Fällen auch nur perspektivische Abbildungen anstelle von echten stereoskopischen Abbildungen.

Auf den folgenden Seiten soll es nun um Techniken zur dreidimensionalen Darstellung und Manipulation von Objekten gehen. Zunächst werden wir die mathematischen Grundlagen erarbeiten und darauf aufbauend einen 3-D-Funktionsplotter wie auch ein kleines CAD-Programm entwickeln.

Anschließend sollen zwei Verfahren zur echten stereoskopischen Darstellung vorgestellt werden, wovon allerdings eines, nämlich das Anaglyphenverfahren, nur für Besitzer eines Farbmonitors praktizierbar ist. Und außerdem müssen Sie auch noch über eine Rot-Grün-Brille verfügen.

## 4.2 Perspektivische Abbildungen

Diese Art von 3-D-Bildern, wie wir sie täglich im Fernsehen und auf Fotos sehen, ist für uns eigentlich die günstigste, erfordert sie doch keinen zusätzlichen technischen Aufwand, sondern nur ein wenig angwewandte Mathematik. Denn bei dieser 'optischen Täuschung' werden sämtliche Eckpunkte eines beispielsweise abzubildenden Hauses auf eine zweidimensionale Fläche projiziert.

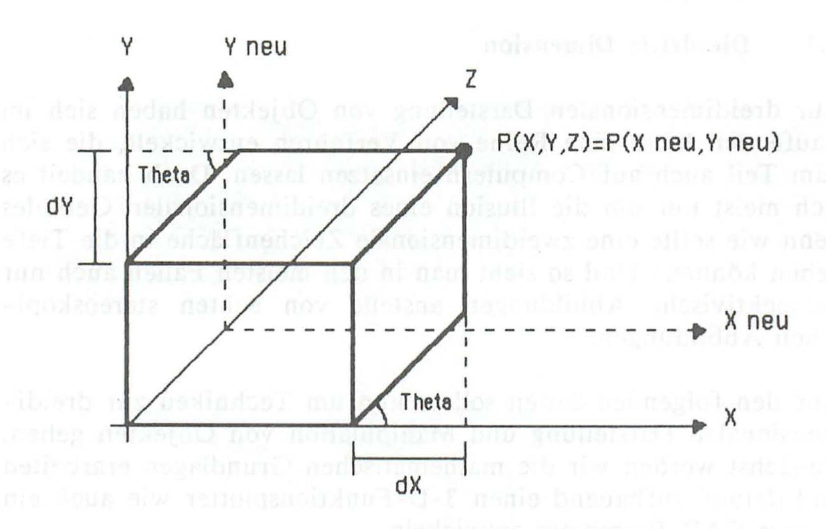



Abb. 39: Die Projektion eines 3-D-Objektes

Dabei werden die Punkte der dritten Achse, der z- oder Raumachse entlang der x-Achse abgebildet. Die Zeichnung macht deutlich, daß zur Darstellung des hinteren rechten Punktes des Würfels eine Koordinatentransformation vorgenommen wurde, und zwar um den Betrag, um den P(x,y,z) von der x-Achse entfernt war, also genau um z. Im neuen Koordinatensystem xneu,yneu läßt der Punkt P(x,y,z) sich dann mit nur zwei Koordinaten angeben: P(xneu,yneu).

Bei einer weiteren Betrachtung der Skizze fällt auf, daß wir mit unserem Wissen bereits in der Lage sein müßten, solche räumlichen Bilder zu erzeugen. Denn wir haben bereits im vorherigen Kapitel eine Formel kennengelernt, die sich auch in diesem Falle anwenden läßt.

So wird die Lage der Raumachse in Bezug auf die x-Achse konstant sein und außerdem von uns gleich zu Beginn festgelegt werden der Winkel den die beiden Achsen somit bilden, ist also bekannt.

Ebenfalls bekannt ist die Länge der Strecke x, und so können wir die Lage des neuen Bildpunktes P(x,y) von P(x,y,z) mit

$$x = X2 + z * COS(THETA)$$

und $$y = Y2 + z * SIN(THETA)$$

angeben.

## 4.3 3-D-Funktionsplotter

Von der Richtigkeit der vorangegangenen Behauptung können wir uns recht schnell anhand unseres Funktionenplotters überzeugen. Im zweidimensionalen Fall wurde eine Variable in Abhängigkeit von einer anderen dargestellt, nun wollen wir die Variable z in Abhängigkeit von x und y berechnen.

Also werden wir um die erste Schleife herum eine zweite aufbauen, welche dann für die verschiedenen y-Werte verantwortlich sein wird. Weiterhin sehen wir für jede der drei Koordinaten einen eigenen Maßstabsfaktor vor, so daß das Bild der Funktion in alle drei Richtungen des Raumes vergrößert und auch verkleinert werden kann:

```
10      'plot3d1.bas
20      '-----------
30      clearw 2 : fullw 2
60      mfx =30 : mfy = 30 : mfz = 5 : cosinus = cos(45) : sinus = sin(45)
160     for y=-6 to 6 step .5
170     for x = -6 to 6 step 0.05
180     gosub 430
210     sx = int(x * mfx + y * mfy * cosinus)
220     sy = int(z * mfz + y * mfy * sinus)
230     linef sx + 320,200-sy,sx + 320, 200-sy
370     next x
```

```
380   next y
390   end
430   z = x*x + y * y
440   return
```

In den Zeilen 210 und 220, in denen die Schirmkoordinaten berechnet werden, erkennen Sie ohne Schwierigkeiten die angegebenen Transformationsformeln. In Zeile 430 wurde als Funktion wieder die Normalparabel eingesetzt, diesmal allerding in räumlicher Form. Es dürfte bei einem Lauf des Programmes somit eine Abbildung ähnlich der Rundung eines Reagenzglases entstehen:

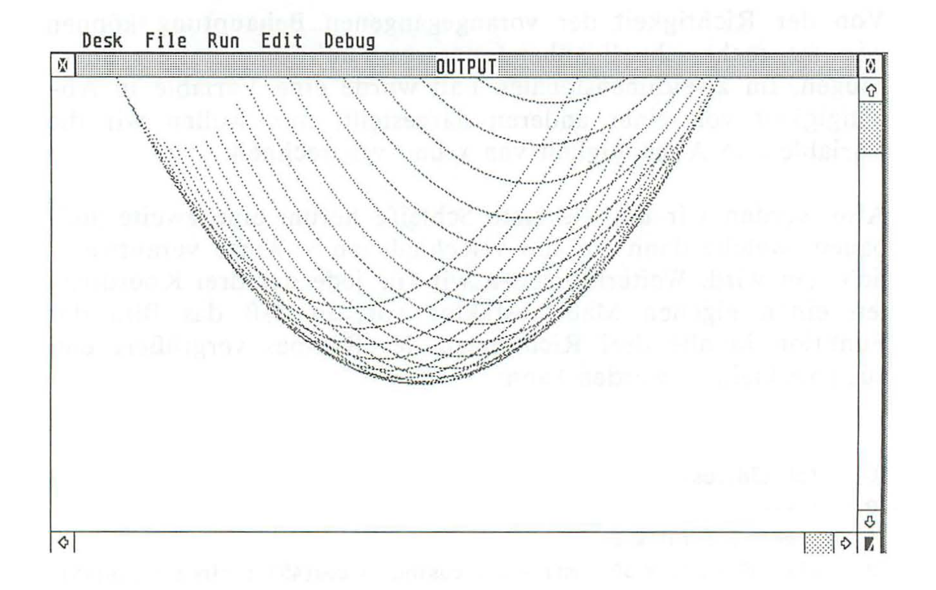



Abb. 40: Die Normalparabel - dreidimensional

Wie dicht die einzelnen Linien beieinander liegen, entscheiden Sie mit den Schrittweiten in den Zeilen 160 und 170. Die Größe der Abbildung können Sie durch Verändern der Werte in 60 festlegen. Damit wäre das wesentliche schon erledigt, und bei

geigneter Wahl der Parameter läßt sich jede Funktion bildschön darstellen. Allerdings wirkt es etwas störend, wenn die Funktionshügel allesamt durchsichtig erscheinen.

### 4.3.1 Die Elimination verdeckter Linien

Starten wir unser Programm noch einmal und achten wir genau auf die Reihenfolge, wie die einzelnen Punkte gesetzt werden.

Zunächst scheint es sich um eine zufällige Verteilung zu handeln, erst nach einiger Zeit läßt sich aus dem Bild der gesetzten Punkte eine Fläche erkennen. Achtet man dann auf die Zeichenrichtung, so stellt man fest, daß die Abbildung punktweise von links nach rechts und schichtweise von vorne nach hinten erstellt wird.

Und hier führt eine einfache Überlegung zum Ziel: Würden wir nämlich die hinten liegenden Teile des Gesamtbildes zuerst zeichnen, dann könnten wir dafür sorgen, daß die jeweils vorderste Schicht alle später dahinterliegen Details verdeckt. Nämlich dadurch, daß wir alle Punkte auf der Linie vom gerade gesetzten Punkt bis zum unteren Bildschirmrand löschen.

Zu diesem Zweck zeichnen wir eine Linie ausgehend von der Plotposition-1

```
320 LINEF SX,SY-1,SX,400
```

- allerdings nicht, ohne zuvor die Hintergrundfarbe zur Zeichenfarbe erklärt zu haben:

```
290 COLOR 1,1,0,1,1
```

die wir anschließend sofort wieder durch eine andere Farbe ersetzen:

```
340 COLOR 1,1,1,1,1
```

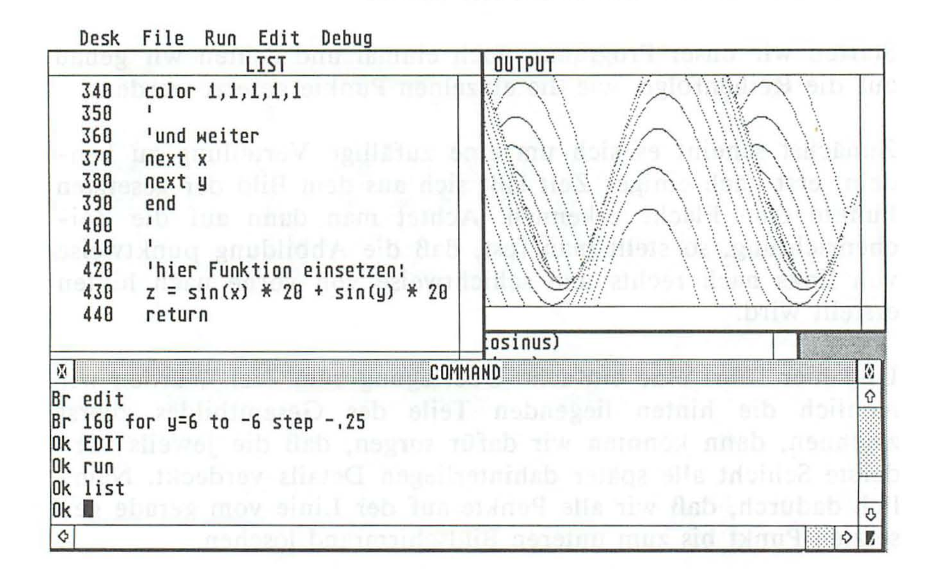



Abb. 41: Hardcopy zum 3-D-Plotter

```
10    ' plot3d mit verdeckten linien
20    ' ----------------------------
30    clearw 2 : fullw 2 : color 1,1,1,1,1
40    '
50    'Vergroesserungsfaktoren
60    mfx =30 : mfy = 30 : mfz = 5
70    '
80    'Sichtwinkel
90    cosinus = cos(45) : sinus = sin(45)
100   y0 = 400   :   ' unterster Bildrand
110   '
```

```
120  'Hilfslinie fuer Schrittweite
130  'linef 1,200,640,200
140  '
150  'Berechnung
160  for y=6 to -6 step -.5
170  for x =-6 to 6 step 0.03
180  gosub 430  :  ' Funktionswert ermitteln
190  '
200  'Werte an Bildschirmkoordinaten anpassen
210  sx = int(x * mfx + y * mfy * cosinus)
220  sy = int(z * mfz + y * mfy * sinus)
230  sx = sx + 320 : sy = 200 - sy :syl = sy +1
240  '
250  'Punkt plotten
260  linef sx,sy,sx,sy
270  '
280  'Zeichenfarbe = Hintergrundfarbe
290  color 1,1,0,1,1
300  '
310  'und verdeckte Linien loeschen
320  linef sx,syl,sx,y0
330  'ab jetzt wieder Zeichenfarbe aktiv
340  color 1,1,1,1,1
350  '
360  'und weiter
370  next x
380  next y
390  end
400  '
410  '
420  'hier Funktion einsetzen:
430  z = sin(x) * 20 + sin(y) * 20
440  return
```

## 4.4 3-D-Darstellung realer Objekte

Naturgemäß besteht der nächste Schritt darin, auf diese Art und Weise auch reale Objekte auf dem Bildschirm darzustellen. In der Tat ist dies mit unseren Formeln auch möglich, doch wird der abzubildende Gegenstand zunächst einmal in ein Zahlenmodell umgesetzt werden müssen, welches dann die Basis für sämtliche Manipulationen darstellt.

Dieser Vorgang könnte so aussehen, daß wir das abzubildende Objekt ausmessen, wobei wir eine Ecke dieses Gegenstandes willkürlich als Nullpunkt bestimmen, und alle anderen markanten Punkte als gemessene Entfernungen zu diesem Punkt angeben.

Unser schon häufig für Manipulationen ähnlicher Art herangezogenes Haus ließe sich dann folgendermaßen beschreiben:

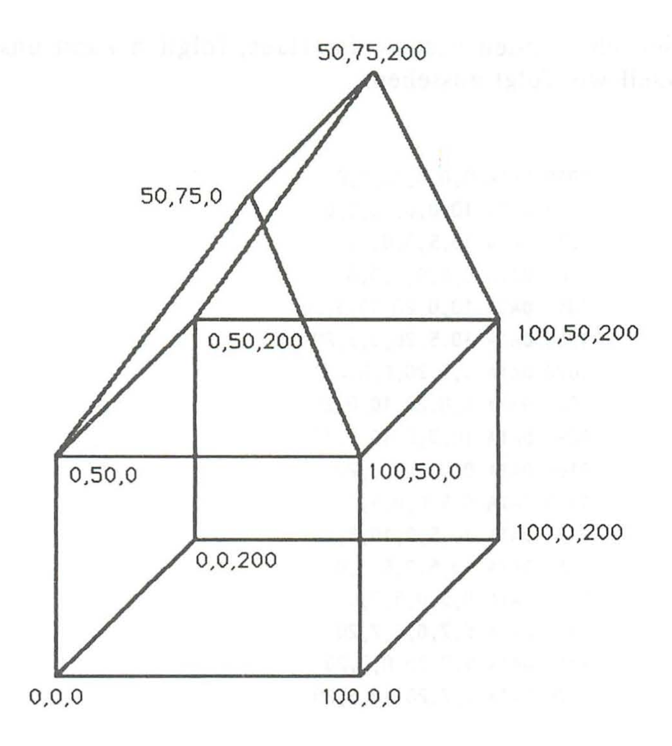



Abb. 42: 3-D-Koordinaten unseres Demo-Hauses

Diese Koordinatentripel müssen dem Rechner nun in irgendeiner Weise zur Bearbeitung angeboten werden, wobei allerdings eine genaue Regelung zu treffen ist, wie die Beziehungen der Punkte untereinander sind. Denn nur die Punkte allein machen noch kein Bild, erst die sie verbindenden Linien lassen uns die Figur erkennen.

Zweckmäßig wäre es daher, von jeder Linie den Anfangs- und Endpunkt anzugeben, denn dadurch erübrigt sich auch die Festlegung einer bestimmten Reihenfolge des Zeichenvorganges.

Außerdem wird das Bild sich leicht editieren lassen, wenn jede Linie für sich verlängert, verkürzt, verschoben oder gelöscht werden kann.

Aus siebzehn Linien besteht das Haus, folglich kann unser Zahlenmodell wie folgt aussehen:

```
1010 DATA 0,0,0,10,0,0
1020 DATA 10,0,0,10,5,0
1030 DATA 10,5,0,0,5,0
1040 DATA 0,5,0,0,0,0
1050 DATA 10,0,20,10,5,20
1060 DATA 10,5,20,0,5,20
1070 DATA 0,5,20,0,0,20
1080 DATA 0,0,20,10,0,20
1090 DATA 10,0,0,10,0,20
1100 DATA 0,0,0,0,0,20
1110 DATA 0,5,0,0,5,20
1120 DATA 10,5,0,10,5,20
1130 DATA 10,5,0,5,7,0
1140 DATA 0,5,0,5,7,0
1150 DATA 5,7,0,5,7,20
1160 DATA 5,7,20,0,5,20
1170 DATA 5,7,20,10,5,20
```

Wenn wir dann noch zuallererst die Anzahl der Linien angeben, können alle Angaben in eine Variablentabelle der Form xvon,yvon,zvon und xnach,ynach,znach eingelesen werden.

```
200 for l=1 to 17
210 read xv(l),yv(l),zv(l),xn(l),yn(l),zn(l)
220 next l
```

Damit befindet sich ein naturgetreues Abbild des Objektes im Speicher, und ist es ebenso naturgetreu dreidimensional.
Bevor es allerdings gezeichnet werden kann, müssen die Werte zunächst auf unsere Bildschirmkoordinaten x und y umgerechnet werden.

Das kann mit den bekannten Transformationsgleichungen geschehen:

```
250 for l=1 to 17
260 sxv(l)=xv(l)*zv(l)*cos(45):sxn(l)=xn(l)+zn(l)*cos(45)
270 syv(l)=yv(l)+zv(l)*sin(45):syn(l)=yn(l)+zn(l)*sin(45)
280 next l
```

Erst dann kann uns nichts mehr daran hindern, das Bild, Linie für Linie, zu zeichnen:

```
320 for l=1 to 17
330 xv=sxv(l):yv=syv(l):xn=sxn(l):yn=syn(l)
340 gosub transline
350 linef xv,yv,xn,yn
360 next l
```

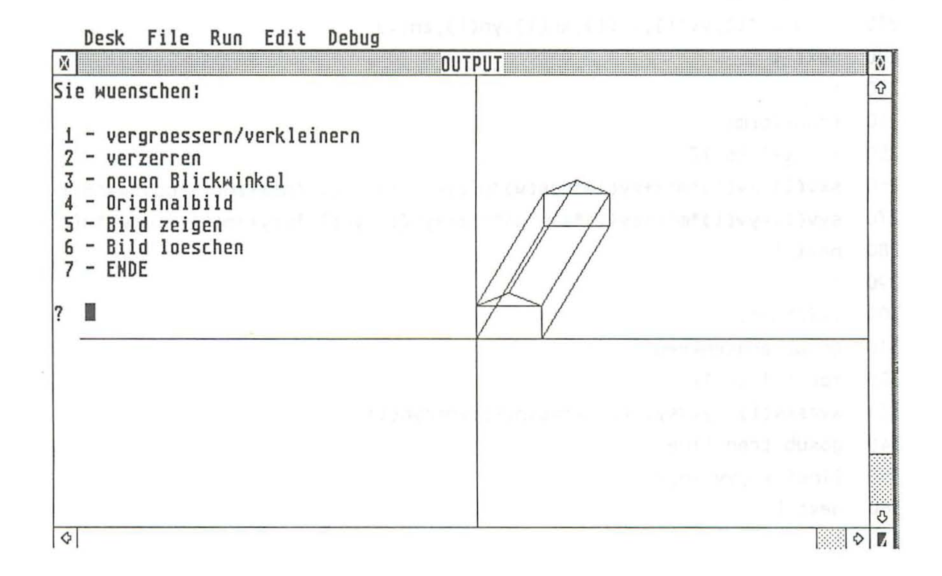



Abb. 43: So präsentiert sich das 3-D-Demo auf dem Schirm

```
10   ' 3d-demo
20   ' Manipulation von 3d-Objekten
30   ' ---------------------------
40   fullw 2:clearw 2
50   '
60   'neues Koordinatensystem
70   xmin = -320 : xmax = 320
80   ymin = -200 : ymax = 200
90   '
100  'Initialisierung
110  read anz
120  dim xv(anz),yv(anz),zv(anz),xn(anz),yn(anz),zn(anz)
130  dim sxv(anz),syv(anz),sxn(anz),syn(anz)
140  '
150  start:
160  mfx=5:mfy=5:mfz=5:w=45
170  '
180  ' Bilddaten einlesen
190  restore : read anz
200  for l=1 to 17
210  read xv(l),yv(l),zv(l),xn(l),yn(l),zn(l)
220  next l
230  '
240  transform:
250  for l=1 to 17
260  sxv(l)=xv(l)*mfx+zv(l)*cos(w)*mfz:sxn(l)=xn(l)*mfx+zn(l)*cos(w)*mfz
270  syv(l)=yv(l)*mfy+zv(l)*sin(w)*mfz:syn(l)=yn(l)*mfy+zn(l)*sin(w)*mfz
280  next l
290  '
300  zeichnen:
310  gosub achsenkreuz
320  for l=1 to 17
330  xv=sxv(l) :yv=syv(l) :xn=sxn(l):yn=syn(l)
340  gosub transline
350  linef xv,yv,xn,yn
360  next l
370  '
```

```
380  menue:
390  gotoxy 0,0:print "Sie wuenschen:":print
400  print " 1 - vergroessern/verkleinern"
410  print " 2 - verzerren"
420  print " 3 - neuen Blickwinkel
430  print " 4 - Originalbild"
440  print " 5 - Bild zeigen"
450  print " 6 - Bild loeschen"
460  print " 7 - ENDE"
470  print:input e
480  if (e<0 or e>7) goto menue
490  on e goto massstab,verzerren,winkel,start,zeichnen,cls,ende
500  goto menue
510  '
520  massstab:
530  gotoxy 0,10:input"Groesser (Zahl > 1) oder kleiner (Zahl 0...1)";mf
540  mfy=mfy+mf:mfz=mfz+mf:mfx=mfx+mf
550  goto transform:
560  '
570  verzerren:
580  gotoxy 0,10:input "In welche Richtung: x=1, y=2, z=3 ";r
590  input" Strecken (Zahl > 1) oder Stauchen (Zahl 0...1)",mf
600  if (r<0 or r>3) then goto verzerren
610  on r goto 620,630,640
620  for l=1 to anz:xv(l)=xv(l)*mf:xn(l)=xn(l)*mf:next l:goto transform
630  for l=1 to anz:yv(l)=yv(l)*mf:yn(l)=yn(l)*mf:next l:goto transform
640  for l=1 to anz:zv(l)=zv(l)*mf:zv(l)=zv(l)*mf:next l:goto transform
650  '
660  winkel:
670  input"Wieviel Grad zwischen der x und der z-Achse ";w
680  goto transform
690  '
700  cls:
710  clearw 2:goto menue
720  '
730  ende:
740  clearw 2:closew 2:end
750  '
```

```
760  ' hier die Bilddaten
770  ' schema: anzahl linien
780  ' linie von (xv,yv,zv) - line nach (xn,yn,zn)
790  '
800  data 17
810  data 0,0,0,10,0,0
820  data 10,0,0,10,5,0
830  data 10,5,0,0,5,0
840  data 0,5,0,0,0,0
850  data 10,0,20,10,5,20
860  data 10,5,20,0,5,20
870  data 0,5,20,0,0,20
880  data 0,0,20,10,0,20
890  data 10,0,0,10,0,20
900  data 0,0,0,0,0,20
910  data 0,5,0,0,5,20
920  data 10,5,0,10,5,20
930  data 10,5,0,5,7,0
940  data 0,5,0,5,7,0
950  data 5,7,0,5,7,20
960  data 5,7,20,0,5,20
970  data 5,7,20,10,5,20
980  '
990  achsenkreuz:
1000 xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
1010 linef xv,yv,xn,yn  : rem x-Achse
1020 xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
1030 linef xv,yv,xn,yn  : rem y-Achse
1040 return
1050 '
1060 transline:
1070 xv = 640 * (xv - xmin) / (xmax - xmin)
1080 yv = 400 * (ymax - yv) / (ymax - ymin)
1090 xn = 640 * (xn - xmin) / (xmax - xmin)
1100 yn = 400 * (ymax - yn) / (ymax - ymin)
1110 return
```

### 4.4.1 Anwendung: CAD

Wie wir gesehen haben, können die Punkte, sobald sie sich erst einmal im Rechner befinden, nach sämtlichen Regeln der Kunst manipuliert werden.

So ist im Prinzip doch auch dieses Haus nichts weiter als ein Shape, es wurde nur in etwas anderer Form dargestellt als in Kapitel 3. Der einzige Unterschied besteht darin, daß dort jeder Punkt, mit Ausnahme des ersten, der Endpunkt einer Linie war, und hier wird zunächst immer erst ihr Beginn angegeben.

Der Umgang mit den Shapes bleibt jedoch auch im Bereich der 3-D-Grafik gleich, egal ob es sich dabei um Vergößerungen, Verkleinerungen, Drehungen oder Verzerrungen handelt.

Wir konnten unser Haus vergrößern, verkleinern und verzerren, was wir allerdings noch nicht tun konnten, ist, uns dem Haus beliebig von irgendeiner Seite zu nähern, also den Betrachterstandpunkt zu verändern.

Vielleicht ahnen Sie es schon: auch für diese Aufgabe gibt es wieder eine Matrix. Doch ist diese in Anbetracht der Daten, die sie verarbeiten muß (Objektbeschreibung xyz, Betrachterstandpunkt xyz), recht aufwendig, so daß ich Ihnen und mir die Herleitung ersparen möchte.

Geben Sie stattdessen einfach das folgende Listing ein und experimentieren Sie mit den unterschiedlichsten Eingabewerten.

```
10    'CADDEMO.BAS
20    'Perspektivisches Zeichnen
30    'Ansicht nach Wahl von allen Seiten
40    'und aus saemtlichen Entfernungen
50    '--------------------------------
60    clearw 2 : fullw 2
70    manz = 20 :'hier maximale Linienzahl festlegen
80    dim xv(manz),yv(manz),zv(manz),xn(manz),yn(manz),zn(manz)
```

```
90     dim xvon(manz),yvon(manz),zvon(manz),xnach(manz),ynach(manz),znach
(manz)
100    '
110    ' bx, by, bz = Winkel des Betrachters zu den Achsen
120    bwx = 0 : bwy = 0 : bwz = 0
130    '
140    ' bsx, bsy, bsz = Koordinaten des Betrachters
150    bsx = 125 : bsy = 100: bsz = 75
160    '
170    neu.start:
180    gosub bild.lesen
190    gosub verschieben
200    gosub matrix
210    gosub verdrehen
220    gosub ansicht
230    gosub zeichnen
240    '
250    menue:
260    gotoxy 0,0:print " 1 - Betrachterstandpunkt "
270    print " 2 - Neigungswinkeld. Betrachters"
280    print:input"Was wollen Sie ändern ";e
290    if (e<0 or e>2) then goto menue
300    on e goto n.s, n.w
310    '
320    n.s:
330    print "Betrachterstandpunkt jetzt:";bsx;bsy;bsz
340    input "Betrachterstandpunkt neu: ";bsx,bsy,bsz
350    goto neu.start
360    '
370    n.w:
380    print "Winkel jetzt:";bx;by;bz
390    input "Winkel neu: ";bx,by,bz
400    goto neu.start
410    '
420    'Transformationsmatrix berechnen
430    matrix:
440    sx=sin(bwx) : cx=cos(bwx)
450    sy=sin(bwy) : cy=cos(bwy)
460    sz=sin(bwz) : cz=cos(bwz)
470    mat(0,0) = sx * sz * sy + cz * cy
```

```
480   mat(0,1) = sx * cz * sy - sz * cy
490   mat(0,2) = sy * cx
500   mat(1,0) = sz * cx
510   mat(1,1) = cz * cx
520   mat(1,2) = -sx
530   mat(2,0) = sx * sz * cy - sy * cz
540   mat(2,1) = sx * cz * cy + sz * sy
550   mat(2,2) = cx * cy
560   return
570   '
580   verschieben:
590   for p=1 to anz
600   xv(p) = xv(p) - bsx : yv(p) = yv(p) - bsy : zv(p) = zv(p) - bsz
610   xn(p) = xn(p) - bsx : yn(p) = yn(p) - bsy : zn(p) = zn(p) - bsz
620   next p
630   return
640   '
650   verdrehen:
660   for p=1 to anz
670   xvon(p) = xv(p) * mat(0,0) + yv(p) * mat(1,0) + zv(p) * mat(2,0)
680   yvon(p) = xv(p)* mat(0,1) + yv(p) * mat(1,1) + zv(p) * mat(2,1)
690   zvon(p) = xv(p)* mat(0,2) + yv(p) * mat(1,2) + zv(p) * mat(2,2)
700   xnach(p) = xn(p) * mat(0,0) + yn(p) * mat(1,0) + zn(p) * mat(2,0)
710   ynach(p) = xn(p) * mat(0,1) + yn(p) * mat(1,1) + zn(p) * mat(2,1)
720   znach(p) = xn(p) * mat(0,2) + yn(p) * mat(1,2) + zn(p) * mat(2,2)
730   next p
740   return
750   '
760   ansicht:
770   for p=1 to anz
780   xvon(p) = xvon(p) * (100/zvon(p)):yvon(p) = yvon(p) * (100/zvon(p)
)
790   xnach(p) = xnach(p) * (100/znach(p)):ynach(p) = ynach(p) * (100/zn
ach(p))
800   next p
810   return
820   '
830   zeichnen:
840   clearw 2
850   for p=1 to anz
```

```
860   linef xvon(p)+140,yvon(p)+96,xnach(p)+140,ynach(p)+96
870   next p
880   return
890   '
900   'Koordinaten des Objektes einlesen
910   bild.lesen:
920   restore
930   read anz
940   for p=1 to anz
950   read xv(p),yv(p),zv(p),xn(p),yn(p),zn(p)
960   next
970   return
980   '
990   'hier die Objektkoordinaten
1000  data 17
1010  data 0,0,0,10,0,0
1020  data 10,0,0,10,5,0
1030  data 10,5,0,0,5,0
1040  data 0,5,0,0,0,0
1050  data 10,0,20,10,5,20
1060  data 10,5,20,0,5,20
1070  data 0,5,20,0,0,20
1080  data 0,0,20,10,0,20
1090  data 10,0,0,10,0,20
1100  data 0,0,0,0,0,20
1110  data 0,5,0,0,5,20
1120  data 10,5,0,10,5,20
1130  data 10,5,0,5,7,0
1140  data 0,5,0,5,7,0
1150  data 5,7,0,5,7,20
1160  data 5,7,20,0,5,20
1170  data 5,7,20,10,5,20
```

### 4.4.2 Die Maus als Eingabegerät

Das letzte Problem, das sich uns nun noch stellt, ist die jeweilige Eingabe der Daten eines Objektes. Hier ist immer eine Menge Vorstellungskraft nötig, um die notwendigen Editiermaßnahmen auf ein Minimum zu beschränken.

Professionelle CAD-Systeme bedienen sich dazu eines speziellen Hilfsgerätes, meist in Form einer mit einem Fadenkreuz versehenen Lupe, an der sich auch noch eine Reihe von Funktionstasten befinden. Dieses Ding wird vielfach ebenfalls Maus genannt, eine Tatsache, die uns nun weiterforschen lassen sollte, ob wir unsere Maus nicht ebenso einsetzen können.

Das GEM-VDI führt bei sämtlichen Mausoperationen auch über deren Position genau Buch und stellt die x- wie auch y-Koordinate bei einem solchen Aufruf im ptsout-Array bereit. Wie wir aber bereits wissen, läßt sich dieses Array auch vom BASIC aus relativ problemlos abrufen. Schreiben wir daher ein kurzes Unterprogramm, welches bei einer Betätigung der linken Maustaste die Mausposition innerhalb der Variablen xm und ym bereitstellt:

```
2010  maus.input:
2020  poke contrl,124
2030  poke contrl+2,0
2040  poke contrl+6,0
2050  vdisys
2060  xm=peek(ptsout)
2070  ym=peek(ptsout+2)
2080  klick=peek(intout)
2090  return
```

Alles, was nun noch zu tun bleibt, ist, aus diesem Aufruf eine Routine zu entwickeln und diese dann auf geeignete Art und Weise in unsere Programme einzubinden - wie es hier bei unserem 3-D-Demonstrationsprogramm geschehen ist.

Nach dem Programmstart werden Sie zunächst dazu aufgefordert, die Maus auf die zu digitalisierende Abbildung zu eichen.

Dazu haben Sie die Maus zuvor mit einem Zeiger versehen - ein kurzes Stück Draht oder eine Stecknadel, die Sie an der Seite oder unten mittels Tesafilm festgeklebt haben -, den Sie nun auf

den äußersten links unten befindlichen Punkt der zu digitalisierenden Abbildung zeigen lassen. Sobald Sie dann die linke Maustaste betätigen, entspricht dieser Punkt der Koordinate (0,0).

Bewegen Sie dann den Zeiger der Maus - aber bitte langsam, denn Sie arbeiten mit BASIC - auf den ersten zu registrierenden Punkt. Betätigen Sie den linken Mausknopf, um die Koordinaten zu speichern. Bewegen Sie den Mauszeiger anschließend zur nächsten Position, und betätigen Sie wiederum die linke Maustaste. Da es sich um 3-D-Koordinaten handelt, müssen Sie zv und zn zusätzlich per Hand und nach Gefühl eingeben - und schon befindet sich die Linie im Speicher Ihres ST.

```
10    ' 3d-demo
20    ' Koordinateneingabe mit der Maus
30    ' ----------------------------
40    fullw 2:clearw 2
50    '
60    'neues Koordinatensystem
70    xmin = -320 : xmax = 320
80    ymin = -200 : ymax = 200
90    '
100   'Initialisierung
110   anz = 100 : ' Platz fuer 100 Linien
120   dim xv(anz),yv(anz),zv(anz),xn(anz),yn(anz),zn(anz)
130   dim sxv(anz),syv(anz),sxn(anz),syn(anz)
140   '
150   start:
160   mfx=5:mfy=5:mfz=5:w=45:l=1
170   gosub eichen
180   gotoxy 0,0:print"Linie: ";l,xv(l);yv(l);xn(l);yn(l)
190   einlesen:
200   gosub maus.input:xv(l)=xm:yv(l)=ym:input "Zv ";zv(l)
210   gosub maus.input:xn(l)=xm:yn(l)=ym:input "Zn ";zn(l)
215   gotoxy 0,0:print"Linie: ";l,xv(l);yv(l);xn(l);yn(l)
220   if zv(l)<>-99 then l=l+1 : goto einlesen
230   anz = l
240   transform:
```

```
250   for l=1 to 17
260    sxv(l)=xv(l)*mfx+zv(l)*cos(w)*mfz:sxn(l)=xn(l)*mfx+zn(l)*cos(w)*mf
z
270    syv(l)=yv(l)*mfy+zv(l)*sin(w)*mfz:syn(l)=yn(l)*mfy+zn(l)*sin(w)*mf
z
280   next l
290   '
300   zeichnen:
310   gosub achsenkreuz
320   for l=1 to 17
330   xv=sxv(l) :yv=syv(l) :xn=sxn(l):yn=syn(l)
340   gosub transline
350   linef xv,yv,xn,yn
360   next l
370   '
380   menue:
390   gotoxy 0,0:print "Sie wuenschen:":print
400   print " 1 - vergroessern/verkleinern"
410   print " 2 - verzerren"
420   print " 3 - neuen Blickwinkel
430   print " 4 - Neu Abtasten"
440   print " 5 - Bild zeigen"
450   print " 6 - Bild loeschen"
460   print " 7 - ENDE"
470   print:input e
480   if (e<0 or e>6) goto menue
490   on e goto massstab,verzerren,winkel,start,zeichnen,cls,ende
500   goto menue
510   '
520   massstab:
530   gotoxy 0,10:input"Groesser (Zahl > 1) oder kleiner (Zahl 0...1)";m
f
540   mfy=mfy+mf:mfz=mfz+mf:mfx=mfx+mf
550   goto transform:
560   '
570   verzerren:
580   gotoxy 0,10:input "In welche Richtung: x=1, y=2, z=3 ";r
590   input" Strecken (Zahl > 1) oder Stauchen (Zahl 0...1)",mf
600   if (r<0 or r>3) then goto verzerren
610   on r goto 620,630,640
```

```
620   for l=1 to anz:xv(l)=xv(l)*mf:xn(l)=xn(l)*mf:next l:goto transform
630   for l=1 to anz:yv(l)=yv(l)*mf:yn(l)=yn(l)*mf:next l:goto transform
640   for l=1 to anz:zv(l)=zv(l)*mf:zv(l)=zv(l)*mf:next l:goto transform
650   '
660   winkel:
670   input"Wieviel Grad zwischen der x und der z-Achse ";w
680   goto transform
690   '
700   cls:
710   clearw 2:goto menue
720   '
730   ende:
740   clearw 2:closew 2:end
750   '
760   ' hier die Bilddaten
770   ' schema: anzahl linien
980   '
990   achsenkreuz:
1000  xv = -300 : yv = 0:xn = 300 : yn = 0 : gosub transline:
1010  linef xv,yv,xn,yn   : rem x-Achse
1020  xv = 0 : yv = -200 : xn = 0 : yn = 200 : gosub transline:
1030  linef xv,yv,xn,yn   : rem y-Achse
1040  return
1050  '
1060  transline:
1070  xv = 640 * (xv - xmin) / (xmax - xmin)
1080  yv = 400 * (ymax - yv) / (ymax - ymin)
1090  xn = 640 * (xn - xmin) / (xmax - xmin)
1100  yn = 400 * (ymax - yn) / (ymax - ymin)
1110  return
1999  '
2000  maus.input:
2010  poke contrl,124
2020  poke contrl+2,0
2030  poke contrl+6,0
2040  vdisys
2050  xm=int((peek(ptsout)+xdiff)*vfm)
2060  ym=int((ydiff-peek(ptsout+2))*vfm)
2070  gotoxy 1,1:print "Mauspos. x/y";xm;ym
2080  klick=peek(intout)
```

```
2090  if klick <> 1 then goto maus.input
2100  print chr$(7):return
2990  '
3000  eichen:
3010  clearw 2: vfm = 1:xdiff=0 : ydiff = 0
3020  print "Bildabtastung mit der Maus:":print
3030  print "Bewegen Sie die Maus so lange, bis ihr Zeiger sich"
3040  print "in der linken unteren Ecke des Bildschirmes befindet." :pri
nt
3050  print "Setzen Sie sie dann vorsichtig so ab,"
3060  print "das der ECHTE Mauszeiger auf die unterste Ecke links"
3070  print "des einzugebenden Bildes zu liegen kommt." : print
3080  print "Betaetigen Sie dann den linken Mausknopf!"
3090  gosub maus.input
3100  xdiff=xm : ydiff=-ym
3110  clearw 2:print "Prima! - Das Bild kann nun abgetastet werden."
3120  print:print"Geben Sie nun einen Vergroesserungs/Verkleinerungs-"
3130  input"faktor ein (keine Manipulation = 1); Empfehlung: 0.3 ";vfm
3140  print:print"Zum Beenden der Eingabe muessen Sie fuer Zv -99 eingeb
en!"
3150  print:input "Kann es losgehen ";e$
3160  clearw 2 : return
```

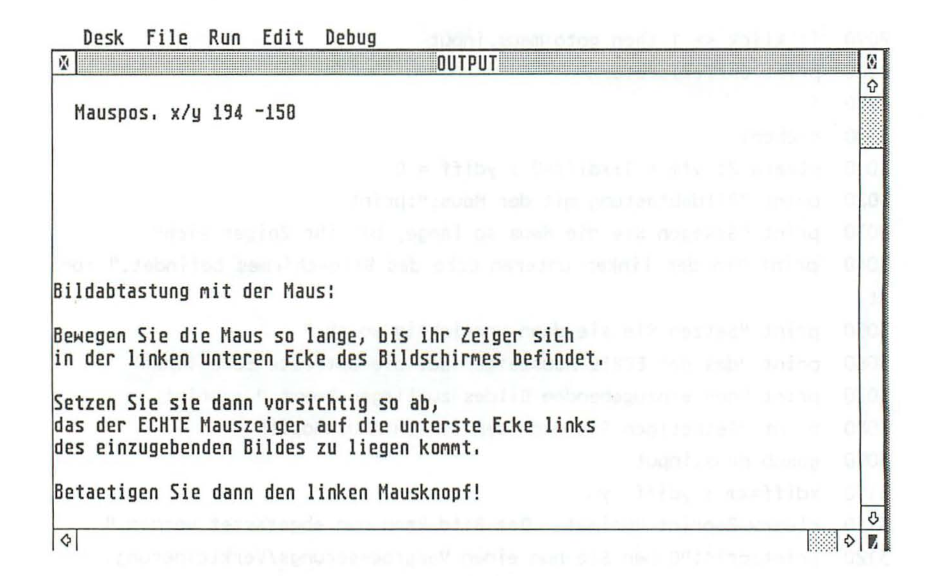



Abb. 44: Koordinateneingabe mit der Maus

### 4.4.3 Echte Stereoskopie

Perspektivische Darstellungen, wie gehabt, sind zwar schon recht schön und gut, doch vermögen sie immer noch nicht, den Eindruck eines realen Objektes - natürliches Sehen - zu vermitteln. Jedem von uns ist klar, wie das räumliche Sehen funktioniert: die beiden Augen nehmen zwei Halbbilder auf, die sich, bedingt durch den Abstand der Augen voneinander, ein wenig unterscheiden. Auf getrennten Wegen werden sie zum Sehzentrum des Gehirns geleitet, dort entsteht das Gesamtbild, welches dann einen räumlichen Eindruck vermittelt.

Das Problem der räumlichen Darstellung ist nun gelöst, sobald einem jeden Auge nur die ihm zustehenden Informationen zukommen. Wir müssen eine strikte Bildtrennung einhalten, d. h., einem jeden Auge muß (auf dem Monitor) das Bild vermittelt werden, daß es aufnehmen würde, wenn ein Mensch vor dem abgebildeten Objekt stünde.

In der Praxis wurde solch ein Verfahren bereits in den Kindertagen der Photografie benutzt. Es wurden von jedem Objekt entweder durch zwei nacheinander mit etwas seitlich versetztem Apparat Aufnahmen gemacht, was bei bewegten Objekten allerdings immer die Tücke desselben zeigte, oder aber es wurden Doppelapparate benutzt, deren Objektive ca. 6.5 cm voneinander entfernt, gleich eingestellt und durch einen gemeinsamen Auslöser betätigt worden waren. Nach dem Vergrößern erhielt der Interessent dann zwei nebeneinanderstehende SW-Fotos, die er in seinen Betrachter einlegen konnte. Dieser Betrachter bestand im wesentlichen aus einer Grundplatte, auf die das Bild aufgelegt wurde, und einem langen Brett, dessen eines Ende die Nasenspitze berührte und dessen anderes Ende sich genau im Zwischenraum zwischen den beiden Bildern befand. Damit war eine vollständige Trennung der Bildinformationen erreicht, und wenn der Betrachter sich genügend anstrengte, war es ihm möglich, ein räumliches Bild wahrzunehmen.

Dieses Verfahren ist natürlich problemlos auf sämtliche Computer übertragbar. Wir müssen nur den Bildschirm als Grundplatte auffassen, unser Gesicht irgendwo davor positionieren und mittels eines Stückes Pappe dafür sorgen, daß das rechte Auge nicht sieht, was fürs linke bestimmt ist (umgekehrt gilt das natürlich auch).

Auf diesen zwei Schirmhälften erzeugen wir dann die entsprechenden Halbbilder:

```
10    'stereo1.bas
20    '----------
30    clearw 2:fullw 2
40    'Halbbild 1
50    gosub breit
60    read anz
70    for l=1 to anz
80    read xv,yv,xn,yn
90    xv=xv * .9 :xn=xn*.9:yv=yv*.9:yn=yn*.9
100   linef xv,yv,xn,yn
110   next
```

```
120   'Halbbild 2
130   read anz
140   for l=1 to anz
150   read xv,yv,xn,yn
160   xv=xv * .9 :xn=xn*.9:yv=yv*.9:yn=yn*.9
170   linef xv+320,yv,xn+320,yn
180   next
190   end
200   breit:
210   poke ptsin,6
220   poke ptsin+2,0
230   poke contrl,16
240   poke contrl+2,1
250   poke contrl+6,0
260   vdisys 16
270   return
280   'frontseite
290   data 10
300   data 40,300,225,320
310   data 225,320,225,210
320   data 225,210,40,190
330   data 40,190,40,300
340   'rechte seite
350   data 225,320,300,250
360   data 300,250,300,100
370   data 300,100,225,210
380   data 225,210,225,320
390   'seite links
400   data 40,190,135,50
410   data 135,50,300,100
420   'frontseite
430   data 10
440   data 60,280,225,320
450   data 225,320,225,210
460   data 225,210,60,170
470   data 60,170,60,280
480   'rechte seite
490   data 225,320,320,255
500   data 320,255,320,105
510   data 320,105,225,210
```

```
520   data 225,210,225,320
530   'seite links
540   data 60,170,150,40
550   'oben
560   data 150,40,320,105
```

Wenn Sie es nun schaffen, genug nach außen zu schielen - denn die Betrachtungsachse sollte senkrecht zum Bildschirm stehen - und die beiden Halbbilder zur Deckung zu bringen, sollten Ihnen der Quader eigentlich recht räumlich erscheinen.

Dies ist jedoch gar nicht so einfach, denn unser Gesichtssinn ist so aufgebaut, daß die beiden Augen immer einen Punkt fixieren; die Sehweise mit den nebeneinanderstehenden Bildern ist daher recht unnatürlich und anstrengend. - Was auch die alten Fotografen schon erkannt hatten: Das Luxusmodell des beschriebenen Stereoskops war nämlich noch mit zwei Linsen ausgestattet, die Erleichterung bringen sollten.

Besser ist es daher, wenn die beiden Halbbilder dicht zusammenstehend abgebildet werden.

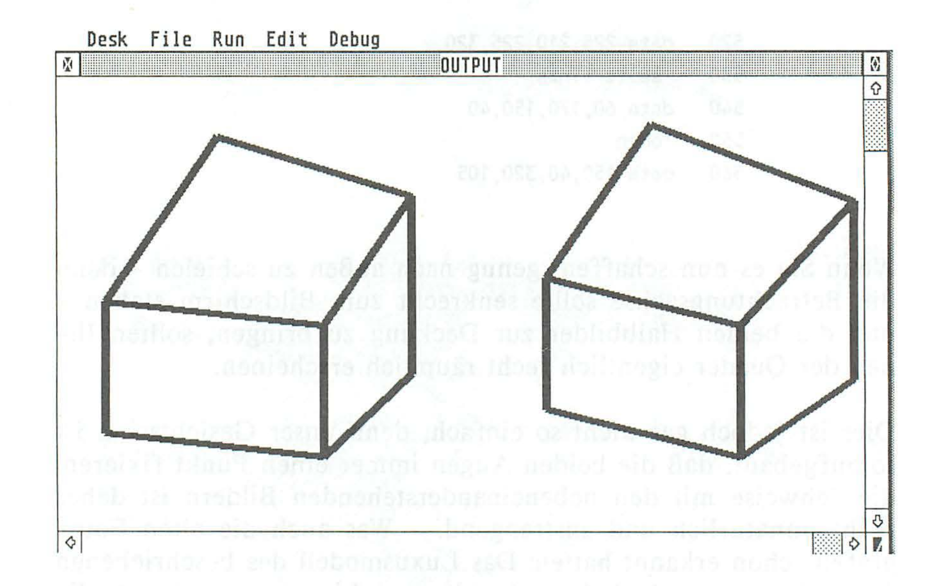



Abb. 45: Die zwei Halbbilder eines Stereobildes

Dann jedoch muß ein anderes Verfahren zur Bildtrennung gewählt werden: In den fünfziger Jahren tauchten dann die ersten 3-D-Kinofilmproduktionen auf, die auf dem Anaglyphenverfahren basierten.

Am Aufnahmeprinzip hatte sich nichts geändert, wohl aber bei der Wiedergabe. Denn von den beiden Halbbildern wurde das eine durch einen Rot- und das andere durch einen Grünfilter projeziert (bzw. waren gleich mit diesen beiden Komplementärfarben eingefärbt), und zwar so, daß sie übereinander zu liegen kamen.

Für das "unbewaffnete" Auge stellte dies natürlich ein Bild mit doppelten Konturen dar, doch wurde dem Betrachter eine entsprechend gefärbte Brille aufgesetzt, so daß auf Grund der verschiedenen Wellenlängen jedem Auge nur die gewünschten Informationen zukamen. Denn die Farbfilter vor den Augen löschten jeweils das eine Bild aus, während das andere weiterhin sichtbar blieb. Ein Verfahren, das durchaus auch auf jeden farbgrafikfähigen Computer anzuwenden ist.

Bevor wir uns jedoch näher mit diesem Thema befassen, sei noch die Bemerkung gestattet, daß sich mit diesem Verfahren natürlich nur SW-Bilder betrachten lassen (die zudem noch eingefärbt erscheinen). Um auch farbige 3-D-Darstellungen betrachten zu können, bedient man sich anstelle der Farbfilter heute zweier Polfilter. Deren Achsen werden dann so ausgerichtet, daß wiederum jedem Auge nur die ihm gehörigen Informationen zukommen. Doch benötigt man dazu auch polarisiertes Licht, weshalb dieses Verfahren für uns kaum in Frage kommen dürfte.

Was Sie unbedingt benötigen, wenn Sie sich mit echten stereoskopischen Darstellungen befassen wollen, ist zunächst einmal ein Farbmonitor. Wer keinen hat, nun, der braucht nicht zu verzweifeln. Zumindest dann nicht, wenn er über ein Fernsehgerät mit Eurobuchse (Scart-) verfügt, denn auch dort können die Rot-, Grün- und Blausignale eingespeist werden.

Weiterhin ist natürlich unbedingt eine Rot/Grünbrille erforderlich. Wer keine hat, der kann sich aus entsprechend gefärbter durchsichtiger Folie (Bastel- oder Fotofachgeschäft) auch eine basteln - notfalls sogar mit anderen Farben.

In jedem Falle sollten Sie dann jedoch einen genauen Abgleich Ihres ST's und Monitors vornehmen.

Lassen Sie folgenden Zweizeiler laufen und rufen Sie das Kontrollfeld auf. Setzen Sie nun Ihre 3-D-Brille auf, und ändern Sie die Farben 2 und 3 so lange, bis Sie, sobald ein Auge zugekniffen ist, jeweils nur einen der Farbbalken wahrnehmen können. (Mit dem linken Auge dürfen Sie nur den linken und mit dem rechten nur den rechten Balken sehen.)

```
10 clearw 2:fullw 2:color 1,1,2,1,1:linef 10,10,10,300
20 color 1,1,3,1,1:linef 20,10,20,300
```

Merken Sie sich diese Einstellung, denn Sie ist typisch für Ihre Kombination Brille/Sichtgerät! Oder besser noch, speichern Sie sie mit der TOS-Option 'Arbeit sichern' auf Ihrer Systemdiskette.

Nun ist alles bereit, um zu wirklichen 3-D-Grafiken zu gelangen. Alles, was Sie nun noch tun müssen, um den schon erwähnten Quader wirklichkeitsgetreu sehen zu können, ist, folgendes Listing einzugeben. Allerdings sollten Sie nicht vergessen, die Brille zu diesem Zweck erst noch einmal abzunehmen.

```
10   'stereo2.bas
20   '-----------
30   clearw 2:fullw 2
40   'Halbbild 1
50   gosub breit
60   read anz : color 1,1,2,1,1
70   for l=1 to anz
80   read xv,yv,xn,yn
90   xv=xv * .9 + 180:xn=xn*.9 + 180:yv=yv*.9:yn=yn*.9
100  linef xv,yv,xn,yn
110  next
120  'Halbbild 2
130  read anz : color 1,1,3,1,1
140  for l=1 to anz
150  read xv,yv,xn,yn
160  xv=xv * .9 + 180:xn=xn*.9 + 180:yv=yv*.9:yn=yn*.9
170  linef xv,yv,xn,yn
180  next
190  end
200  breit:
210  poke ptsin,3
220  poke ptsin+2,0
230  poke contrl,16
240  poke contrl+2,1
250  poke contrl+6,0
260  vdisys 16
270  return
280  'frontseite
```

```
290   data 10
300   data 40,300,225,320
310   data 225,320,225,210
320   data 225,210,40,190
330   data 40,190,40,300
340   'rechte seite
350   data 225,320,300,250
360   data 300,250,300,100
370   data 300,100,225,210
380   data 225,210,225,320
390   'seite links
400   data 40,190,135,50
410   data 135,50,300,100
420   'frontseite
430   data 10
440   data 60,280,225,320
450   data 225,320,225,210
460   data 225,210,60,170
470   data 60,170,60,280
480   'rechte seite
490   data 225,320,320,255
500   data 320,255,320,105
510   data 320,105,225,210
520   data 225,210,225,320
530   'seite links
540   data 60,170,150,40
550   'oben
560   data 150,40,320,105
```

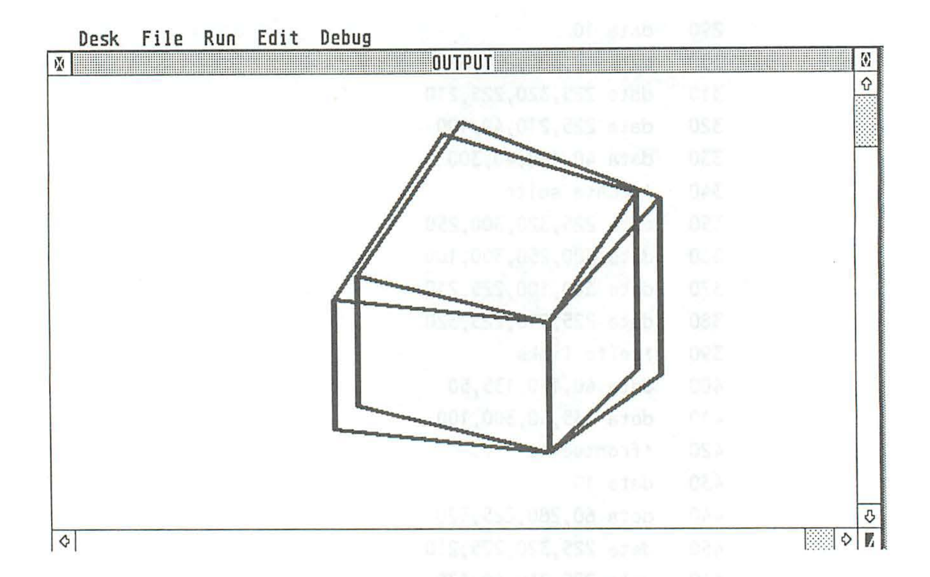



Abb. 46: Hardcopy eines Stereobildes auf dem ST

Die Frag, die Sie sich nun stellen, lautet sicher: "Wie gelange ich denn nun wieder an geeignete Vorlagen?" und ist durchaus berechtigt. Schließlich wollen Sie nicht stundenlang herumprobieren - was allerdings auch zu sehr interessanten Effekten führen kann.

Nun, der einfachste und genaueste Weg wäre der, daß Sie die gewünschten Objekte fotografieren - und zwar zweimal, wie bereits beschrieben mit etwas versetzter Kamera. Dabei sollten Sie sich dann keinesfalls auf den Basisabstand von
6 cm beschränken, sondern auch größere Distanzen ausprobieren. Meist wird der beabsichtigte Effekt dadurch natürlich noch viel deutlicher.

Sobald die entwickelten Bilder dann aus der Entwicklungsanstalt kommen, müssen Sie nur noch die geeigneten Linien auf das Koordinatensystem des ATARI ST übertragen, wobei Ihnen die zuvor entwickelte Eingaberoutine sicherlich eine große Hilfe sein wird.

Einen letzten Tip zu diesem Thema möchte ich noch an all diejenigen richten, die neben der Computerei auch noch ein Fotolabor betreiben. Entwickeln und vergrößern Sie Ihre Negative extrem hart oder benutzen Sie Konturfilm (bzw. bedienen Sie sich einer Solarisationstechnik). Dann erhalten sie im günstigsten Fall bereits eine Vorlage, auf der nur noch die wichtigsten Linien sichtbar sind, und die entsprechend leicht zu digitalisieren ist.

# 5. Kapitel

# Tricks, Tips
# ... und noch mehr Grafik

## 5.1 Tips & Tricks

Im Rahmen dieses Kapitels möchte ich auf einige Fragen eingehen, deren Antworten früher oder später auch für Sie von Interesse sein werden:

1. Wie kann eine Applikation feststellen, ob ein hoch- oder niedrigauflösender Monitor an den ST angeschlossen ist?

2. Wie schreibe ich ein C-, Modula- oder Pascalprogramm, das in sämtlichen Auflösungsstufen funktionsfähig bleibt?

3. Wie viele Farbstifte stehen mir zur Verfügung und wie kann ich deren Farben einstellen?

4. Wie kann ich verschiedene Farben auf dem SW-Monitor simulieren, sprich: mit Graustufen arbeiten?

5. Wie kann ich komplette Bilder speichern und laden?

6. Wozu verschiedene Schreibmodi - und wie stelle ich sie um?

### 5.1.1 Feststellen der Auflösung

Wie Sie wissen, sendet der SW-Monitor ein Signal aus, welches dem ATARI ST vorschreibt, nur HiRes zuzulassen. Eigentlich gedacht als Schutzmaßnahme für die üblichen Standardmonitore, ist diese Tatsache für die meisten von uns des öfteren doch recht ärgerlich.

Kommen wir doch immer wieder in den Besitz von Programmen, die nach dem Starten eine Alarmbox folgenden Inhaltes präsentieren: 'Diese Anwendung unterstützt nur den Farbmonitor! - Abbruch.'

Warum das Programm nicht laufen kann, wissen wir, nämlich wegen des Gebrauchs der Rasterkoordinaten anstelle von NDC. Und auch wie die Software an die Kenndaten des Monitors kommt, wurde bereits erwähnt.

So befinden sich unter den 45 Daten in intout nach dem Aufruf Open Workstation eben auch die Maximalwerte für x und y und eine Angabe über die Anzahl der von der Ausgabestation unterstützten Farben (in intout(13)). Ist ein Monochrom-Monitor angeschlossen, stehen nur Hintergrund- und Vordergrundfarbe bereit (0 und 1), der entsprechende Rückgabewert wird demnach zwei sein.

Sobald intout(13) > 2 ist, muß es sich um einen Farbmonitor handeln. Zur Abwechslung könnten wir dann einen Form-Alert auslösen und nur den SM124 als Arbeitsstation zulassen. Wie es beispielsweise bei den später folgenden Listings zu den Apfelmännchen geschehen ist, die speziell für ein Koordinatensystem von 640 x 400 Punkten geschrieben wurden.

### 5.1.2 Low-, Mid- und High-Resolution

Damit ein Programm in allen drei Beriebsarten funktionsfähig ist, kann es zum einen mit den normalisierten Koordinaten (32k x 32k) arbeiten. Sollte der damit verbundene Geschwindigkeitsverlust kritisch sein, kann die Applikation die notwendigen Umrechnungen auch selbst vornehmen - auf ähnliche Art und Weise, wie wir es auch in den vorangegangenen Kapiteln getan haben.

Denken wir nur einmal an unsere TRANSPOINT- und TRANSLINE-Routinen, die Punktkoordinaten von einem System in ein anderes umrechneten. Im Falle der Auflösungsstufen des ST ist sogar noch eine einfachere Transformation ausreichend: ergibt sich Mid-Res doch aus Hi-Res durch zwei! D. h., daß sich sämtliche Umrechnungen auf eine Multiplikation oder Division mit zwei beschränken, abhängig davon, für welche Betriebsart wir das Programm entwickelt haben.

In allen Fällen müssen jedoch xmax und ymax der aktuellen Arbeitsstation bekannt sein, Werte, die sich nach Open Workstation in intout(0) und intout(1) befinden.

### 5.1.3 Farbeinstellung

Reine Anwender können sich auf den Gebrauch der Schieber des Kontrollfeldes beschränken. Dort lassen sich - nach Anklicken des entsprechenden Kästchens - bis zu sechzehn Farben durch Mischen von Rot, Grün und Blau definieren, wobei die Skala jeweils von null bis sieben geht.

Programmierer unter BASIC können sich zur Auswahl dieser Farbstifte des Schlüsselwortes color t,f,l,x,x bedienen, wobei t für die Textfarbe, für die Füllfarbe und l für die Linienfarbe steht.

Ein LOGO-Programmierer kann sich seine aktuelle Farbe aus dem Spektrum der 512 möglichen zusammenmischen. Hier ersetzt die Anweisung

```
SETPAL stift (farbliste)
```

die Schieber des Kontrollfeldes. Die Farbliste legt den R-, G- und B-Anteil der Farbe von Stift Nummer stift fest; der Regelbereich erstreckt sich hier über einen Bereich von 0 bis 1000. SETPAL 1 (1000 0 0) wird Stift 1 demgemäß tiefrot tönen.

Analog dazu ist die Vorgehensweise unter GEM (also für Compilersprachen-Benutzer), müssen die drei Farbanteile eines jeden Stiftes hier doch im Array RGB.IN(3) abgelegt werden:

```
int rgb_in(3);

rgb_in(0) = 1000;
rgb_in(1) = 0;
rgb_in(2) = 0;
vs_color(handle,stift,rgb_in);
```

Mit dieser Anweisungsfolge definiert ein C-Programmierer zunächst ebenfalls die Farbe tiefrot. Anschließend wird diese dem gewünschten Stift zugewiesen. Wäre stift beispielsweise gleich 0, würde der Hintergrund entsprechend eingefärbt werden.

Ist allerdings ein SW-Monitor angeschlossen, so werden alle Einstellungen hinfällig. Denn dann gibt es nur die Farben 0,0,0 und 1000,1000,1000. Immerhin kann man den Monitor damit auf inverse Darstellung umschalten.

Ein Hinweis noch zum Schluß: Sie müssen die runden Klammern in obigen Beispielen durch eckige, die wir aus drucktechnischen Gründen hier nicht darstellen konnten, ersetzen.

### 5.1.4 Graustufen auf dem SW-Monitor

In den allermeisten Fällen wird man sich wohl eine möglichst hohe Auflösung des Bildes wünschen, in einigen Fällen wäre jedoch eine mehrfarbige Darstellung angebrachter. Auch als SM124-Besitzer muß man in solchen Fällen nicht mit dem Schicksal hadern, kann man doch ersatzweise mit Graustufen arbeiten - wenn man gewillt ist, dafür eine niedrigere Auflösung in Kauf zu nehmen.

Arbeitet man beispielsweise nur noch mit Koordinaten im Bereich zwischen 0,0 und 320,200, dann entsprechen einem jeden dieser Punkte vier Pixel auf dem HiRes-Bildschirm. Durch entsprechende Gruppierungen können wir dann fünf Graustufen nutzen:

```
weiß:      - -
           - -
```

```
grau1:      # -
            - -

grau2:      # -
            - #

grau3:      # #
            # -

schwarz:    # #
            # #
```

Auch für die Nutzung dieses Tricks finden Sie bei den Apfelmännchen wieder ein Beispiel.

### 5.1.5 Bilder speichern

Manchmal verspürt man auch den Wunsch, ein Bild auf der Diskette zu speichern - besonders dann, wenn eine relativ große Zeitspanne verstrichen ist, ehe es vollendet wurde.

Denken wir doch nur einmal an unseren 3-D-Plotter. Wollen wir eine solide aussehende Grafik erzeugen, d.h. haben wir die Schrittweiten entsprechend klein gewählt, dann muß man sich schon in Geduld üben.

Als zweckmäßig erweist es sich da, den Bildschirminhalt nach Beendigung des Plotvorganges abzuspeichern, ein Vorgang, der sich problemlos erledigen läßt. Denn seit dem ersten Kapitel dieses Buches ist uns der Speicherbereich bekannt, innerhalb dessen die Grafik bitweise aufgebaut wird. Und diesen ganzen

Block können wir mit der Anweisung BSAVE auf die Diskette kopieren (und später mit BLOAD wieder von dort laden):

```
bsave "screen.bin",&H78020,32000

bload "screen.bin"
```

Mit dieser Anweisungsfolge haben Sie zunächst ein Bild von einem 520ST bzw. 260ST unter dem Namen screen.bin auf der Diskette gespeichert. Die Parameter hinter dem Namen waren dabei Angaben über Startadresse und Länge des zu speichernden Blockes.

Wird mit bload außer dem Namen nicht auch noch eine Ladeadresse angegeben, dann wird der Speicherauszug während des Ladevorganges an die Originaladressen geladen.

Die Auswirkung in der Praxis: Auf einem Megajack wäre nichts neues zu sehen. Glücklicherweise kann jedoch optional eine Ladeadresse angegeben werden, so daß Bilder auch ausgetauscht werden können.

### 5.1.6 Der Schreibmodus

Der ST kann beim Beschreiben des Bildschirmes die neu zu setzenden Punkte mit den alten logisch verknüpfen, wozu er sich der Bool'schen Operatoren AND, OR, NOT und XOR bedient. Verknüpft werden dabei das zu zeichnende Objekt und dessen Farbe mit der Farbe eines bereits gesetzten Punktes.

Ohne nun in die Theorie der Bool'schen Algebra abzuschweifen läßt sich sagen, daß durch logische Operationen vier verschiedenene Schreibmodi erzielt werden können:

1. Der Replace-Modus entspricht der normalen Betriebsart, d.h. jeder Punkt wird unabhängig von allen anderen Faktoren gesetzt.

2. Ist der Transparent-Modus aktiviert, sind alle Objekte durchsichtig, altes und neues Objekt werden gemischt, da die neuen Punkte zusätzlich zu den alten gesetzt werden.

3. Wurden die Ausgabefunktionen auf den XOR-Modus umgestellt, dann werden nur dort Punkte gesetzt, wo bisher noch die Hintergrundfarbe zu sehen war. Wo alte und neue Punkte zur Deckung kommen, wird der zuvor gesetzte Punkt gelöscht und die Hintergrundfarbe sichtbar.

4. In der Betriebsart Reverse Transparent werden alle Punkte gesetzt, die zuvor gelöscht waren und umgekehrt.

Zum Setzen der gewünschten Schriftart bietet das VDI die Funktion Set Writing Mode an. Der Aufruf in C lautet:

```
vswr_mode(handle, modus)
```

Beispiele für das Setzen der verschiedenen Schriftarten entnehmen Sie bitte dem Listing Animationsgrafik, von dem Sie sowohl eine BASIC- als auch eine Modula-2-Version in diesem Kapitel finden können.

### 5.1.7 Ausfüllen mit Mustern

Mit dem Ausfüllen von Figuren haben wir uns bereits befaßt; allerdings haben wir dabei nur geschlossene Flächen mit einer Farbe zulaufen lassen. GEM kennt jedoch eine Anzahl von Mustern, die für einen ebensolchen Einsatz gedacht sind und die ein kurzes LOGO-Programm hier vorstellen soll:

```
TO FILLSHOW
HT CS
SETLINE [1 1 1]
```

```
(LOCAL "XB "YB "W "Z)
MAKE "GFILL "TRUE
MAKE "W PI / 6
MAKE "Z (180 / PI)
FILLRUF 2 1 0
ATARI
FILLRUF 2 13 0
CARO
FILLRUF 3 1 0
AB
MAKE "GFILL "FALSE
STOP
END

TO FILLRUF :SELECT :BEGIN :LAUF
IF :LAUF > 11 [STOP]
SETFILL (LIST :SELECT (:BEGIN + :LAUF) 1)
MAKE "XB (5 * SIN (:W * :Z))
MAKE "YB (5 * COS (:W * :Z))
IF :LAUF < 6 [FILLZEICH :LAUF (-1)]
IF :LAUF > 5 [FILLZEICH :LAUF 1]
MAKE "W (:W + (PI / 3))
FILLRUF :SELECT :BEGIN (:LAUF + 1)
END

TO FILLZEICH :DEG :PLACE
ARC (LIST (:PLACE * 153 + :XB) (0 + :YB) 145 (:DEG * 60) (:DEG * 60 +
60))
STOP
END

TO ATARI
SETFILL [4 1 1]
FILL
WAIT 50
CS
END
```

```
TO CARO
PPROP "GRAPHICS ".FPT [0 0 128 448 992 2032 4088 8188 16382 8188 4088
2032 992 448 128 0]
SETFILL [4 1 1]
FILL
WAIT 50
CS
END

TO AB
PPROP "GRAPHICS ".FPT [0 0 0 0 3704 8060 6924 6924 13176 13176 13068
25356 25468 25464 0 0]
SETFILL [4 1 1]
FILL
WAIT 50
CS
END

TO WAIT :ZEIT
IF :ZEIT < 1 [STOP]
WAIT :ZEIT - 1
END

MAKE "GFILL "FALSE
```

Diese Patterns können selbstverständlich auch unter C oder BASIC genutzt werden, im letzten Fall sind dafür die Parameter vier und fünf der Anweisung COLOR zuständig. Und zwar wählt der vierte Wert nach Color den Füllstil, und der fünfte Wert das eigentliche Füllmuster aus.

## 5.2 Noch mehr Grafik

Zum Abschluß des Teiles dieses Buches, der sich mit Grafik befaßt, möchte ich Ihnen noch einige Grafikprogramme ohne Theorie und lange Erläuterungen vorstellen, die sicherlich auch Ihr Interesse wecken.

### 5.2.1 Animationsgrafik

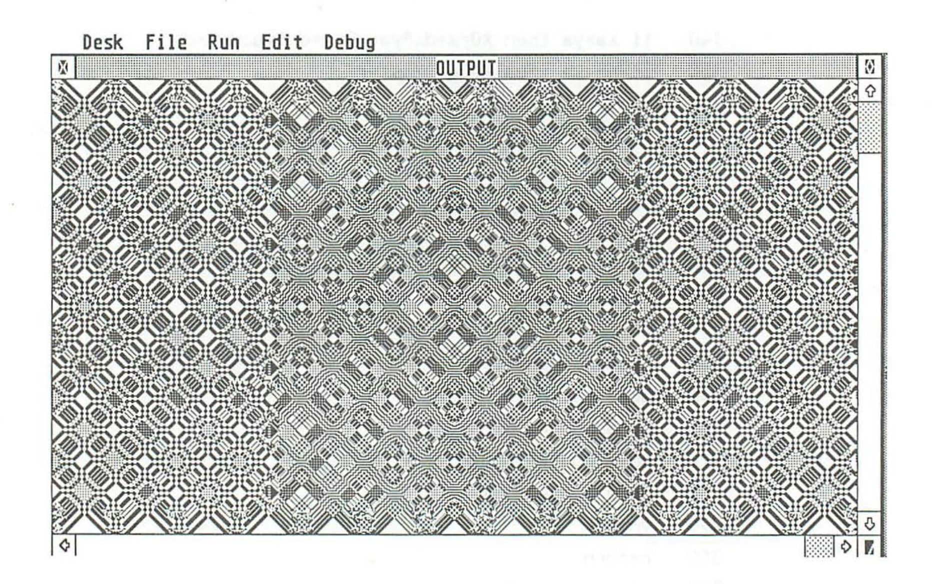
Animationsgrafiken waren bislang speziellen Rechnern vorbehalten, erfordern sie doch eine extrem schnelle Berechnung der Koordinaten und müssen die einzelnen Bildelemente ebenfalls möglichst schnell dargestellt und wieder gelöscht werden.

Nun ist eine 8-Mhz-68000-CPU nicht eben langsam. Und kann man sich dann noch eines XOR-Schreibmodus bedienen, um Details zu löschen, lassen sich bereits respektable Ergebnisse erzielen, wie das Modula-Listing Kaleidoskop beweist. (BASIC-Programmierer finden auch eine BASIC-Version; natürlich ist diese nicht ganz so schnell und der Effekt daher nicht so deutlich).

Übrigens, eine Betätigung der rechten Maustaste führt zu neuen Mustern, ein Klick links zum Desktop.

```
10    defint d,x,y
20    fullw 2
30    Neues.Muster:
40    gosub Replace.Modus
50    clearw 2
60    gosub XOR.Modus
70    gosub Linienbreite
80    x=0:y=0:dx=1:dy=1
90    xm=310+int(rnd*305):ym=174+int(rnd*170)
100   repeat:
110   if dx<0 then xa=x else xa=xm-x
120   if dy<0 then ya=y else ya=ym-y
130   if xa<ya then x0=x+dx*xa:y0=y+dy*xa:dx=-dx
```

```
140   if xa>ya then x0=x+dx*ya:y0=y+dy*ya:dy=-dy
150   if xa=ya then Neues.Muster
160   linef x,y,x0,y0
170   linef 615-x,y,615-x0,y0
180   linef x,344-y,x0,344-y0
190   linef 615-x,344-y,615-x0,344-y0
200   x=x0:y=y0
210   if x<>0 or y<>0 then repeat
220   goto repeat
230   rem ----------
240   Replace.Modus:
250   poke intin,1
260   poke contrl,32
270   poke contrl+2,0
280   poke contrl+6,1
290   vdisys
300   return
310   rem ----------
320   XOR.Modus:
330   poke intin,3
340   poke contrl,32
350   poke contrl+2,0
360   poke contrl+6,1
370   vdisys
380   return
390   rem ----------
400   Linienbreite:
410   poke ptsin,1+2*int(rnd*5)
420   poke ptsin+2,0
430   poke contrl,16
440   poke contrl+2,1
450   poke contrl+6,0
460   vdisys
470   return
```



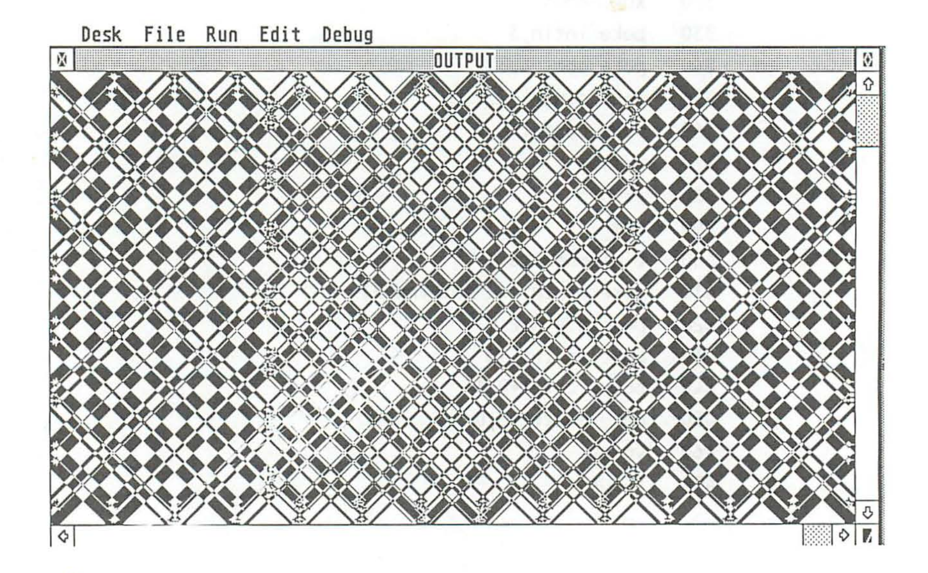




Abb. 47: Hardcopy zum Programm MUSTER.BAS

```
MODULE Muster;

FROM VDIControls IMPORT OpenVirtualWorkstation,
                        CloseVirtualWorkstation,
                        ClearWorkstation;
FROM VDIOutputs  IMPORT PolyLine;
FROM VDIInputs   IMPORT HideCursor;
FROM VDIAttribs  IMPORT SetWritingMode;
FROM GEMVDIbase  IMPORT VDIWorkInType,VDIWorkOutType;
FROM AESGraphics IMPORT GrafHandle,GrafMouseKeyboardState;

VAR handle:INTEGER;
    In:VDIWorkInType;
    Out:VDIWorkOutType;

(*=====================================================================*)

MODULE RandomNumbers;
EXPORT Random;
CONST M  = 100000000;
      m1 = 10000;
      b  = 31415821;
VAR seed : LONGCARD;

PROCEDURE Random ( maxvalue : LONGCARD ) : CARDINAL;

  PROCEDURE Multiply ( p, q : LONGCARD ) : LONGCARD;
  VAR p0, p1, q0, q1 : LONGCARD;
  BEGIN
    p1 := p DIV m1;
    p0 := p MOD m1;
    q1 := q DIV m1;
    q0 := q MOD m1;
    RETURN (((p0*q1+p1*q0) MOD m1) * m1 + p0*q0) MOD M;
  END Multiply;

BEGIN
  seed := (Multiply (seed, b) + 1) MOD M;
  RETURN CARDINAL((((seed DIV m1) * maxvalue) DIV m1) MOD 65536);
END Random;
```

```
BEGIN (* MODULE *)
  seed := 349867;
END RandomNumbers;

(*=====================================================================*)

PROCEDURE Linie(x1,y1,x2,y2:INTEGER);
VAR xyArray:ARRAY [0..3] OF INTEGER;
BEGIN
  xyArray[0]:=x1;
  xyArray[1]:=y1;
  xyArray[2]:=x2;
  xyArray[3]:=y2;
  PolyLine(handle,2,xyArray);
END Linie;

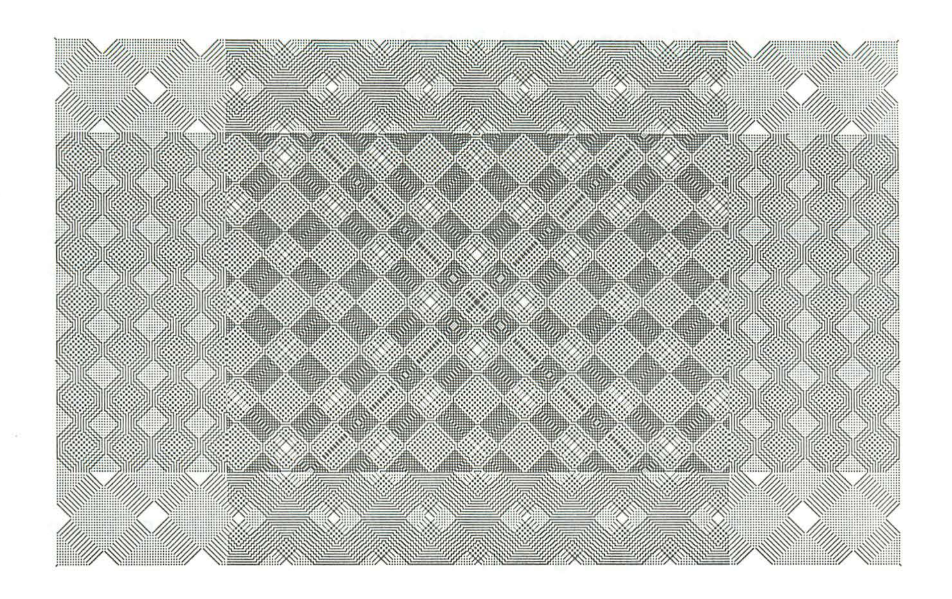
PROCEDURE Init;
VAR i,dummy:INTEGER;
BEGIN
  handle:=GrafHandle(dummy,dummy,dummy,dummy);
  FOR i:=0 TO 9 DO In[i]:=1 END;
  In[10]:=2;
  OpenVirtualWorkstation(In,handle,Out);
  HideCursor(handle);
  ClearWorkstation(handle);
  dummy:=SetWritingMode(handle,3);
END Init;

PROCEDURE Ende;
BEGIN
  CloseVirtualWorkstation(handle);
END Ende;

PROCEDURE Show;
VAR x,y,x0,y0,
    dx,dy,xabs,yabs,
    xmax,ymax,xrand,yrand,
    dummy,klick: INTEGER;
```

```
BEGIN
  xrand:=Out[0];
  yrand:=Out[1];
  REPEAT
    ClearWorkstation(handle);
    x:=0;
    y:=0;
    dx:=1;
    dy:=1;
    xmax:=(xrand DIV 2)+INTEGER(Random(LONGCARD(xrand DIV 2)));
    ymax:=(yrand DIV 2)+INTEGER(Random(LONGCARD(yrand DIV 2)));
    REPEAT
      IF dx<0 THEN xabs:=x ELSE xabs:=xmax-x END;
      IF dy<0 THEN yabs:=y ELSE yabs:=ymax-y END;
      IF xabs<yabs THEN
        x0:=x+dx*xabs;
        y0:=y+dy*xabs;
        dx:=-dx;
      ELSIF xabs>yabs THEN
        x0:=x+dx*yabs;
        y0:=y+dy*yabs;
        dy:=-dy;
      ELSE
        x0:=x+dx*xabs;
        y0:=y+dy*yabs;
        dx:=-dx;
        dy:=-dy;
      END;
      Linie(x,y,x0,y0);
      Linie(xrand-x,y,xrand-x0,y0);
      Linie(x,yrand-y,x0,yrand-y0);
      Linie(xrand-x,yrand-y,xrand-x0,yrand-y0);
      x:=x0;
      y:=y0;
      GrafMouseKeyboardState(dummy,dummy,klick,dummy);
    UNTIL (klick<>0) OR (x=0) AND (y=0);
  UNTIL klick=2;
END Show;
```

```
BEGIN
  Init;
  Show;
  Ende;
END Muster.
```



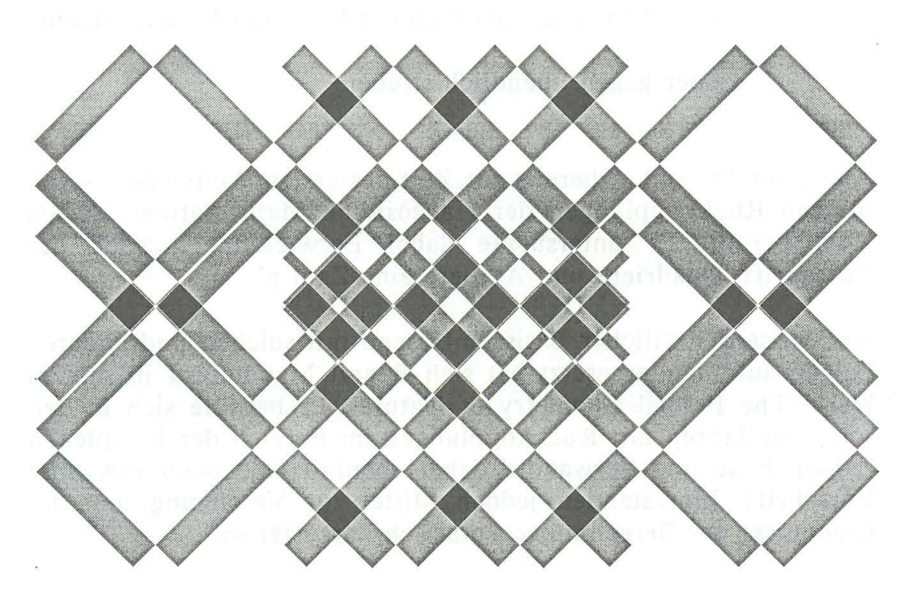


Abb. 48: Hardcopy zum Programm MUSTER.MOD

### 5.2.2 Von Apfelmännern und Fractals

Apfelmännchen - eine Vereinigung von Kunst und Mathematik, handelt es sich dabei doch ebenfalls um ein Funktions-Plotprogramm, wenn auch um ein ganz spezielles.

Normalerweise ist uns eine Reihe von Werten gegeben, auf die wir irgendeine Funktion anwenden und so deren Funktionswerte erhalten. In diesem Falle soll aber auf den Funktionswert wiederum dieselbe Funktion angewandt werden. Solch eine Rückkopplung kann zu folgenden Ergebnissen führen:

Entweder

- wird die Folge der Funktionswerte gegen einen Punkt konvergieren,
- zwischen mehreren Punkten hin- und herschwanken,
- oder gegen unendlich streben.

Als erster hat sich - bereits zur Zeit des ersten Weltkrieges - mit solchen Rückkopplungen der französische Mathematiker Gaston Julia befaßt. Er untersuchte dabei Entwicklungen nach der Vorschrift 'Quadriere und Addiere eine Zahl p'.

Auf wissenschaftlicher Basis umfassend mit solchen und anderen Folgen auseinandergesetzt hat sich Benoit Mandelbrot in seinem Werk 'The Fractal Geometry of Nature'. Er befaßte sich in den achtziger Jahren mit Rückkopplungen im Bereich der komplexen Zahlenebene und verwandte dabei ebenfalls die oben erwähnte Vorschrift. Ihm standen jedoch Mittel zur Verfügung, um die Ergebnisse der Berechnungen grafisch darzustellen.

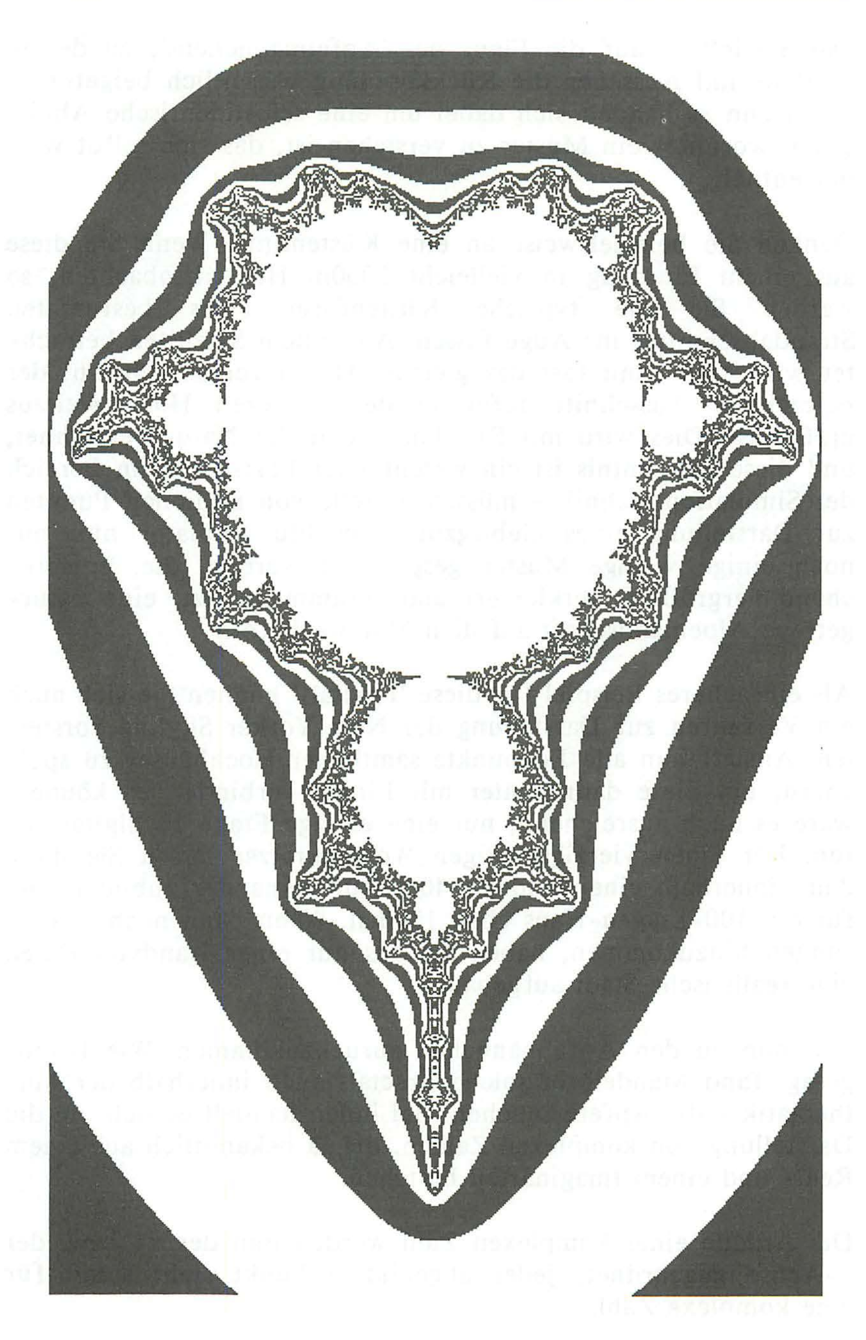


Abb. 49: Das Ur-Apfelmännchen

Dabei stieß er auf die Figur des 'Apfelmännchens', zu dessen Aufbau und Aussehen die Rückkopplung wesentlich beigetragen hat. Denn es handelt sich dabei um eine selbstidentische Abbildung, worunter ein Muster zu verstehen ist, das sich selbst wieder enthält.

Denken Sie beispielsweise an eine Küstenlinie. Wenn Sie diese aus einem Flugzeug in vielleicht 2000m Höhe beobachten, so werden Sie die typische Küstenform eines bestimmten Strandabschnittes ins Auge fassen. Aus einem Satelliten betrachtet wird sich dann fast das gleiche Abbild zeigen, obwohl der betrachtete Ausschnitt aufgrund der größeren Höhe weitaus größer ist. Dies wird mit Regelmäßigkeit der Natur bezeichnet, und diese Erkenntnis ist ein wesentlicher Fortschritt im Bereich der Simulationstechnik - müssen anstelle von millionen Punkten zur Darstellung eines Gebirgzuges im Flugsimulator nun nur noch einige wenige Muster gespeichert werden, die, entsprechend vergrößert, verkleinert und zusammengsetzt, eine naturgetreue Alpenlandschaft auf dem Monitor liefern.

Als einfacheres Beispiel für diese 'Fractals' können Sie sich auch ein Verfahren zur Darstellung der New Yorker Skyline vorstellen. Anstatt nun alle Eckpunkte sämtlicher Hochhäuser zu speichern, um diese dann später mit Linien verbinden zu können, wäre es auch ausreichend, nur eine einzige Etage zu digitalisieren. Für einen vierzigstöckigen Wolkenkratzer lassen Sie diese dann innerhalb einer Schleife 40mal übereinander abbilden und für ein 100-Etagen-Haus eben 100mal. Wenn dann noch Skalierungen hinzukommen, haben Sie aus nur einer Handvoll Daten eine realistische Stadt aufgebaut.

Um nun zu den Apfelmännchen zurückzukommen: Wie bereits gesagt fand Mandelbrot solche Fractals auch innerhalb der Mathematik - die Apfelmännchen. Bei ihnen handelt es sich um die Darstellung von komplexen Zahlen, die ja bekanntlich aus einem Real- und einem Imaginärteil bestehen.

Die Anteile einer komplexen Zahl werden nun der x- bzw. der y-Achse zugeordnet, jeder abgebildete Punkt steht somit für eine komplexe Zahl.

Um dann zu einer Funktionsreihe zu gelangen, wird von einem Anfangswert die komplexe Zahl C abgezogen. Die Differenz - selbst wieder eine komplexe Zahl - wird quadriert; von dem Quadrat wird wiederum C subtrahiert - usw.

Es ergibt sich nun, daß die so berechneten Funktionswerte irgendwann gegen unendlich gehen, womit uns ein Vergleichskriterium bekannt ist, um die Rechnung abbrechen zu lassen; weitere Berechnungen würden schließlich zu keinem anderen Ergebnis mehr führen. Doch bis dieser Punkt erreicht ist, werden meist zahlreiche Rechnungen (Subtraktion, Quadrieren) notwendig sein, bei denen das Ergebnis zunächst immer wieder zwischen kleineren und größeren Werten hin- und herschwankt.

Aus diesem Grunde setzen wir außerdem eine maximale Rechentiefe fest; ist sie erreicht worden, soll die Berechnung ebenfalls abgebrochen und der Punkt geplottet werden.

Dieses Plotten wird nun - zumindest solange nur ein SW-Monitor angschlossen ist - einfach aus dem Setzen dieses Punktes bestehen.

Sobald aber ein Farbmonitor angeschlossen ist und mehrere Farben zur Verfügung stehen, können mittels einer einfachen Erweiterung phantastische Farbgrafiken erstellt werden. Bestimmend für die jeweilige Farbe soll dabei die erreichte Rechentiefe sein. Die Punktfarbe wird also mit TIEFE MODULO ANZAHLFARBEN festgelegt werden.

Wie kommen Sie nun zu den verschiedensten Apfelmännchen?

Fertigen Sie sich dazu zunächst die Hardcopy eines Ur-Apfelmännchens an. Dazu geben Sie die ersten der in eckigen Klammern ausgegebenen Beispielwerte ein.

Das heißt, mit den Real-Anteilen, die ja auf der x-Achse abgebildet werden, legen Sie die Länge der x-Achse fest. Und mit den Imaginär-Anteilen bestimmen Sie entsprechend y.

Die unterschiedlichsten Apfelmännchen sind jeweils Vergrößerungen von Teilausssschnitten dieses Ausgangsbildes. Alles, was Sie zu solch einem 'Zooom in die Mandelbrot-Menge', wie die Vorgehensweise auch schon bezeichnet worden ist, tun müssen, ist, zur Vergrößerung vielversprechender Stellen deren x- und y-Koordinaten zu ermitteln und einzugeben.

Es entsteht eine Vergrößerung eben dieses Ausschnittes, dessen Lage wiederum genau bekannt ist.

Und nun erleben Sie selbst die Auswirkungen der Rückkopplung, denn das Verfahren setzen Sie am neuen Bild fort ... und erhalten das nächste Bild ... Sie wählen einen Ausschnitt, und ...

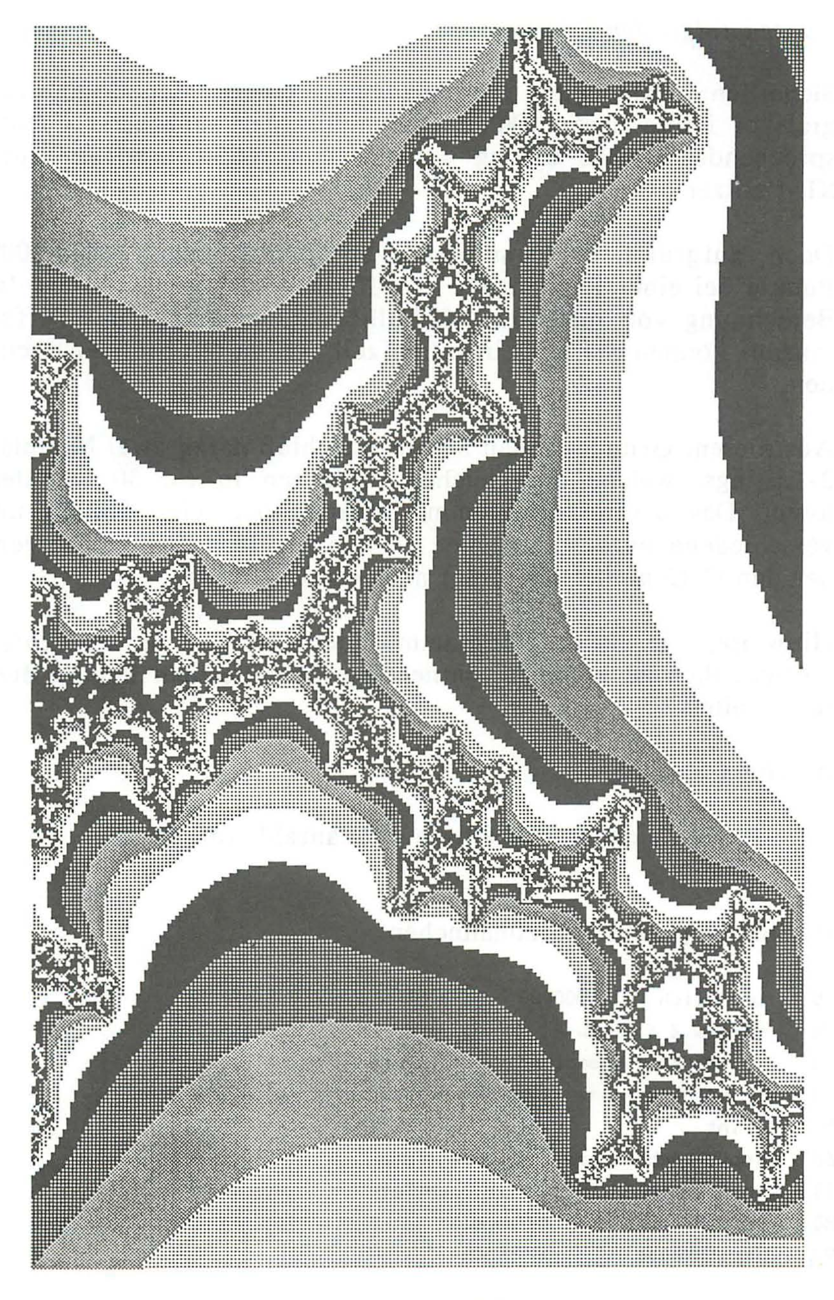


Abb. 50: Apfelmännchen im Bereich 0.0, 0.2, -1.06, -0.9

*Zu den folgenden Listings:*

Sicherlich werden diese Apfelmännchen einen jeden Computergrafiker interessieren. Daher finden Sie hier zunächst ein entsprechendes BASIC-Listing - denn über BASIC verfügt jeder ST-Besitzer.

Doch aufgrund der umfangreichen Berechnungen (128 000 Punkte bei einer Rechentiefe von 10 zu berechnen erfordert die Berechnung von mehr als 1.2 Millionen Quadraten und Differenzen) können Sie mit einer Laufzeit von ca. 10 Stunden rechnen.

Aus diesem Grunde finden Sie im Anschluß daran zwei Modula-2-Listings, welche die gleichen Aufgaben in ca. 30 Minuten lösen. Das zweite Programm benutzt vier Graustufen, um verschiedene Farben darstellen zu können. Und für denjenigen, der den C-Compiler hat, folgt noch ein C-Listing.

Hinweise, wie Sie das Programm ändern müssen, um zu echten Farbgrafiken zu kommen, entnehmen Sie bitte dem Beginn dieses Kapitels.

Beachten Sie dabei:

Zeichenfarbe = tiefe mod anzahl_farben

BASIC-Programm Apfelmännchen

```
10    unendlich = 100000000
20    clearw 2 : fullw 2
30    print "Apfelmaennchen - Juliamengen - Mandelbrotmenge"
40    print "=============================================="
50    print
60    input"Rechentiefe    ";rechentiefe
70    input"Zeichentiefe   ";zeichentiefe
80    input"Breite         ";breit
90    input"Hoehe          ";hoch
```

```
100   input"re-min (xmin) ";remin
110   input"re-max (xmax) ";remax
120   input"im-min (ymin) ";immin
130   input"im-max (ymax) ";immax
140   clearw 2
150   dreal = (remax - remin) / (breit -1)
160   dimag = (immax - immin) / (hoch - 1)
170   for iimag=0 to hoch
180   for ireal=0 to breit
190   lreal = remin + ireal * dreal
200   limag = immin + iimag * dimag
210   re = lreal
220   im = limag
230   tiefe = 0
240   xx = re * re : yy = im * im
250   im = 2 * re * im - limag
260   re = xx - yy - lreal
270   tiefe = tiefe + 1
280   if((tiefe=rechentiefe) or ((xx + yy) > unendlich)) then 290 else 2
40
290   if(tiefe<rechentiefe)then gosub 500
300   next ireal
310   next iimag
320   'plotten
330   if (tiefe>zeichentiefe) then linef ireal,iimag,ireal,iimag else go
sub 600
340   return
350   if (tiefe mod 2) then linef ireal,iimag,ireal,iimag
360   return
```

## Modula-Programm Apfelmännchen

```
MODULE Apfel;

FROM VDIControls IMPORT OpenVirtualWorkstation,
                        CloseVirtualWorkstation,
                        ClearWorkstation;
FROM VDIOutputs  IMPORT PolyMarker;
FROM VDIInputs   IMPORT ShowCursor,HideCursor,SampleMouseButton;
FROM GEMVDIbase  IMPORT VDIWorkInType,VDIWorkOutType;
FROM AESForms    IMPORT FormAlert;
FROM AESGraphics IMPORT GrafHandle;
FROM TextIO      IMPORT WriteString,WriteLn,ReadInt,ReadReal;

VAR handle:INTEGER;
    In:VDIWorkInType;
    Out:VDIWorkOutType;

PROCEDURE Punkt(x,y:INTEGER);
VAR xyArray:ARRAY [0..1] OF INTEGER;
BEGIN
  xyArray[0]:=x;
  xyArray[1]:=y;
  PolyMarker(handle,1,xyArray);
END Punkt;

PROCEDURE Init;
VAR i,dummy:INTEGER;
BEGIN
  handle:=GrafHandle(dummy,dummy,dummy,dummy);
  FOR i:=0 TO 9 DO In[i]:=1 END;
  In[10]:=2;
  OpenVirtualWorkstation(In,handle,Out);
  IF Out[13]>2 THEN
    dummy:=FormAlert(1,'[3][Nur in Schwarz/Weiss!][Ende]');
    HALT;
  END;
  HideCursor(handle);
END Init;
```

```
PROCEDURE Ende;
BEGIN
  CloseVirtualWorkstation(handle);
END Ende;

PROCEDURE Zeichnen;
CONST unendlich=1.0E08;
VAR   breit,hoch,ireal,iimag,
      dummy,klick,choice,Tiefe,
      Rechentiefe,Zeichentiefe:INTEGER;
      remin,remax,immin,immax,
      dreal,dimag,lreal,limag,
      xx,yy,re,im:REAL;

  PROCEDURE Eingaben;
  BEGIN
     WriteString('Apfelmännchen  -  Juliamengen  -  Mandelbrotmenge)');
WriteLn;
           WriteString('===============================================');
WriteLn;
  WriteString('Rechentiefe................[ca. 10..20] ? ');
  ReadInt(Rechentiefe); WriteLn;
  WriteString('Zeichentiefe...............[ca.  5..15] ? ');
  ReadInt(Zeichentiefe); WriteLn;
  WriteString('Bildbreite...................[max. 640] ? ');
  ReadInt(breit); WriteLn;
  WriteString('Bildhoehe....................[max. 400] ? ');
  ReadInt(hoch); WriteLn;
  WriteString('Z-Re-Min [zB:  -0.75  0.0   1.1   1.6 ] ? ');
  ReadReal(remin); WriteLn;
  WriteString('Z-Re-Max [zB:   2.25  0.2   1.27  1.8 ] ? ');
  ReadReal(remax); WriteLn;
  WriteString('Z-Im-Min [zB:  -1.2  -1.06 -0.35 -0.05] ? ');
  ReadReal(immin); WriteLn;
  WriteString('Z-Im-Max [zB:   1.2  -0.9  -0.25  0.05] ? ');
  ReadReal(immax); WriteLn;
  dreal:=(remax-remin)/FLOAT(CARDINAL(breit-1));
  dimag:=(immax-immin)/FLOAT(CARDINAL(hoch-1));
  END Eingaben;
```

```
BEGIN
  REPEAT
    ClearWorkstation(handle);
    Eingaben;
    ClearWorkstation(handle);
    iimag:=0;
    REPEAT
      ireal:=0;
      REPEAT
        lreal:=remin+FLOAT(CARDINAL(ireal))*dreal;
        limag:=immin+FLOAT(CARDINAL(iimag))*dimag;
        re:=lreal;
        im:=limag;
        Tiefe:=0;
        REPEAT
          xx:=re*re;
          yy:=im*im;
          im:=2.0*re*im-limag;
          re:=xx-yy-lreal;
          Tiefe:=Tiefe+1;
        UNTIL (Tiefe>=Rechentiefe) OR ((xx+yy)>unendlich);
        IF Tiefe<Rechentiefe THEN
          IF (Tiefe>Zeichentiefe)
          THEN Punkt(ireal,iimag)
          ELSE IF ODD(Tiefe)
               THEN Punkt(ireal,iimag) END;
          END;
        END;
        ireal:=ireal+1;
        SampleMouseButton(handle,klick,dummy,dummy);
      UNTIL (klick=1) OR (ireal>=breit);
      iimag:=iimag+1;
    UNTIL (klick=1) OR (iimag>=hoch);
    WHILE klick<>1 DO SampleMouseButton(handle,klick,dummy,dummy); END;
    ShowCursor(handle,0);
    choice:=FormAlert(1,
                    '[1][>> APFELMÄNNCHEEN <<|aus DATA BECKERS|Grafik &
Sound][Weiter|Ende]');
    HideCursor(handle);
  UNTIL choice=2;
```

```
END Zeichnen;

BEGIN
  Init;
  Zeichnen;
  Ende;
END Apfel.
```

## Modula-Programm in Graustufen

```
MODULE Apfel2;
(* Version 2 mit Graustufen von 0=Weiss bis 4=Schwarz *)

FROM VDIControls IMPORT OpenVirtualWorkstation,
                        CloseVirtualWorkstation,
                        ClearWorkstation;
FROM VDIOutputs  IMPORT PolyMarker;
FROM VDIInputs   IMPORT ShowCursor,HideCursor,SampleMouseButton;
FROM GEMVDIbase  IMPORT VDIWorkInType,VDIWorkOutType;
FROM AESForms    IMPORT FormAlert;
FROM AESGraphics IMPORT GrafHandle;
FROM TextIO      IMPORT WriteString,WriteLn,ReadInt,ReadReal;

VAR handle:INTEGER;
    In:VDIWorkInType;
    Out:VDIWorkOutType;

PROCEDURE Punkt(x,y,Graustufe:INTEGER);
VAR xyArray:ARRAY [0..3] OF INTEGER;
BEGIN
  x:=x+x; y:=y+y; (* 640 mal 400 *)
  IF Graustufe=0  (* 0=Weiss 4=Schwarz *)
  THEN RETURN;
  ELSIF Graustufe<>2
  THEN xyArray[0]:=x; xyArray[1]:=y;
       PolyMarker(handle,1,xyArray); END;
  IF Graustufe>1
  THEN xyArray[0]:=x+1; xyArray[1]:=y;
       xyArray[2]:=x;   xyArray[3]:=y+1;
```

```
      PolyMarker(handle,2,xyArray); END;
  IF Graustufe>3
  THEN xyArray[0]:=x+1; xyArray[1]:=y+1;
      PolyMarker(handle,1,xyArray); END;
END Punkt;

PROCEDURE Init;
VAR i,dummy:INTEGER;
BEGIN
  handle:=GrafHandle(dummy,dummy,dummy,dummy);
  FOR i:=0 TO 9 DO In[i]:=1 END;
  In[10]:=2;
  OpenVirtualWorkstation(In,handle,Out);
  IF Out[13]>2 THEN
    dummy:=FormAlert(1,'[3][Nur in Schwarz/Weiss!][Ende]');
    HALT;
  END;
  HideCursor(handle);
END Init;

PROCEDURE Ende;
BEGIN
  CloseVirtualWorkstation(handle);
END Ende;

PROCEDURE Zeichnen;
CONST unendlich=1.0E06;
VAR   breit,hoch,ireal,iimag,
      dummy,klick,choice,Tiefe,
      Rechentiefe,Zeichentiefe:INTEGER;
      remin,remax,immin,immax,
      dreal,dimag,lreal,limag,
      xx,yy,re,im:REAL;

  PROCEDURE Eingaben;
  BEGIN
  WriteString('Apfelmänchen - Juliamengen - Mandelbrotmenge)'); WriteLn;
  WriteString('==============================================='); WriteLn;
  WriteString('Rechentiefe................[ca. 10..99] ? ');
  ReadInt(Rechentiefe); WriteLn;
```

```
  WriteString('Zeichentiefe...............[ca.  1..33] ? ');
  ReadInt(Zeichentiefe); WriteLn;
  WriteString('Bildbreite..................[max. 320] ? ');
  ReadInt(breit); WriteLn;
  WriteString('Bildhoehe...................[max. 200] ? ');
  ReadInt(hoch); WriteLn;
  WriteString('Z-Re-Min [zB:  -0.75  0.0   1.1   0.14] ? ');
  ReadReal(remin); WriteLn;
  WriteString('Z-Re-Max [zB:   2.25  0.2   1.27  0.18] ? ');
  ReadReal(remax); WriteLn;
  WriteString('Z-Im-Min [zB:  -1.2  -1.06 -0.35 -1.05] ? ');
  ReadReal(immin); WriteLn;
  WriteString('Z-Im-Max [zB:   1.2  -0.9  -0.25 -1.02] ? ');
  ReadReal(immax); WriteLn;
  Zeichentiefe:=(Zeichentiefe DIV 5)*5+3; (* nach Grau3 kommt Schwarz *)
  dreal:=(remax-remin)/FLOAT(CARDINAL(breit-1));
  dimag:=(immax-immin)/FLOAT(CARDINAL(hoch-1));
  END Eingaben;

BEGIN
  REPEAT
    ClearWorkstation(handle);
    Eingaben;
    ClearWorkstation(handle);
    iimag:=0;
    REPEAT
      ireal:=0;
      REPEAT
        lreal:=remin+FLOAT(CARDINAL(ireal))*dreal;
        limag:=immin+FLOAT(CARDINAL(iimag))*dimag;
        re:=lreal;
        im:=limag;
        Tiefe:=0;
        REPEAT
          xx:=re*re;
          yy:=im*im;
          im:=2.0*re*im-limag;
          re:=xx-yy-lreal;
          Tiefe:=Tiefe+1;
        UNTIL (Tiefe>=Rechentiefe) OR ((xx+yy)>unendlich);
```

```
        IF Tiefe<Rechentiefe THEN
          IF (Tiefe>Zeichentiefe)
          THEN Punkt(ireal,iimag,4)
          ELSE Punkt(ireal,iimag,Tiefe MOD 5) END;
        END;
        ireal:=ireal+1;
        SampleMouseButton(handle,klick,dummy,dummy);
      UNTIL (klick=1) OR (ireal>=breit);
      iimag:=iimag+1;
    UNTIL (klick=1) OR (iimag>=hoch);
    WHILE klick<>1 DO SampleMouseButton(handle,klick,dummy,dummy); END;
    ShowCursor(handle,0);
    choice:=FormAlert(1,
                  '[1][>> APFELMÄNNCHEN <<|aus DATA BECKERS|Grafik &
Sound][Weiter|Ende]');
    HideCursor(handle);
  UNTIL choice=2;
END Zeichnen;

BEGIN
  Init;
  Zeichnen;
  Ende;
END Apfel2.
```

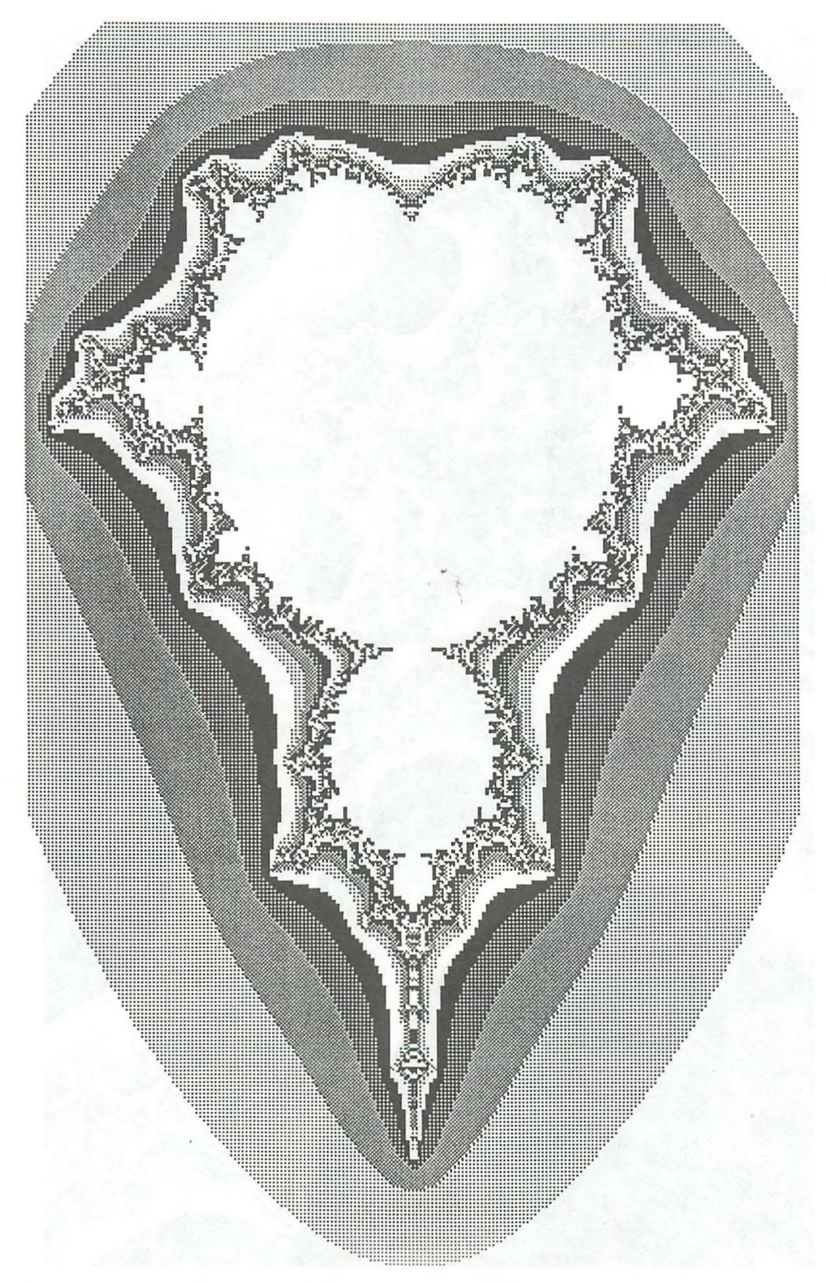


Abb. 51: Das Ur-Apfelmännchen in Graustufen

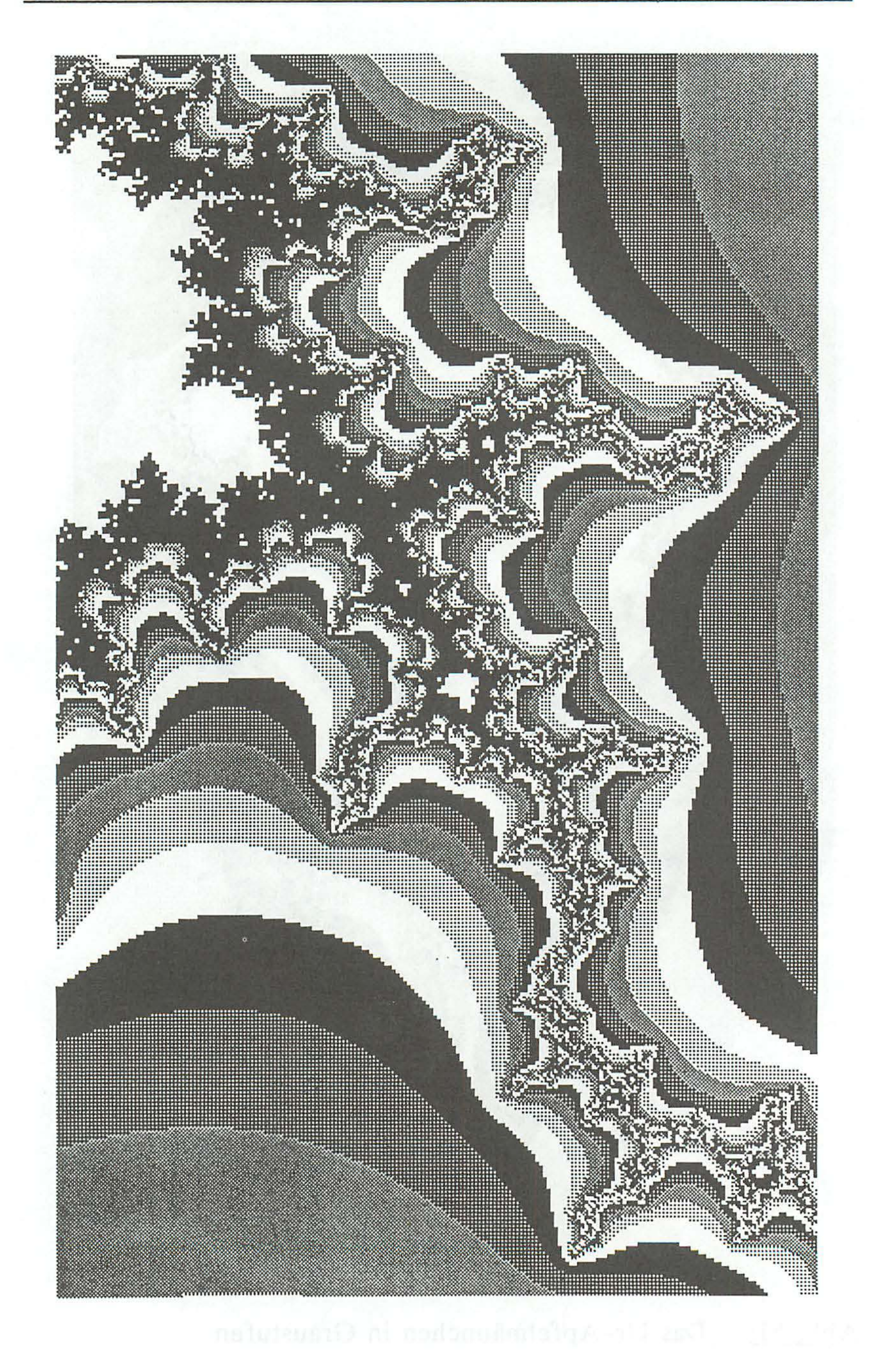


Abb. 52: Apfelmännchen im Bereich 1.1, 1.27, -0.35, -0.25

## C-Programm Apfelmännchen

```
/****************************************************************************/
/*                    APFELMAENNCHEN - JULIAMENGEN                          */
/*          fuer Monochrome Monitor & Hoechstaufloesung                     */
/*                    aus: ST Grafik & Sound                                */
/*                         DATA BECKER                                      */
/*                         V1 - Dez.84                                      */
/****************************************************************************/
                              /* GLOBALE VARIABLEN                          */
int contrl[12];
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];
                              /* GEMVDI & AES GLOBALS                       */
int handle,i;
int pxyarray[1];
int int_in[11];
int int_out[57];
int width, height;
int dummy;                    /* fuer alles unnoetige ...                   */
int klick;                    /* fuer Mausclick                             */
int choice = 0;               /* fuer Nummer des gewaehlten Buttons         */
int ende;
                              /* PROGRAMMVARIABLEN                          */
int i_imag, i_real,hoch, breit;
int rechentiefe, zeichentiefe, tiefe;

float re_min, re_max, im_min, im_max;
float d_real, d_imag, l_real, l_imag;
float xx, yy, re, im;
float unendlich=1.0E08;
```

```
                                /*  ARBEITSSTATION OEFFNEN                 */
open_vwork()
{
int i;
    for (i = 1; i <10; i++){
      int_in[i] = 1;            /* init int_in array: linetype, farbe,  */
      }                         /* fillstyles usw.                      */
    int_in[10] = 2;             /* benutze RC - Koordinaten             */
    v_opnvwk(int_in, &handle, int_out); /* jetzt kanns losgehen ... */
}

                                /* HAUPTPROGRAMM                          */
main()
{
   appl_init();
   handle=graf_handle(&width,&height,&dummy,&dummy);
   open_vwork();
      v_clrwk(handle);
      eingaben();
      v_hide_c(handle);
      v_clrwk(handle);
      d_real=(re_max - re_min) / (breit - 1.0);
      d_imag=(im_max - im_min) / (hoch - 1.0);
      i_imag = 0;
      do{
         i_real = 0;
            do{
               l_real = re_min + i_real * d_real;
               l_imag = im_min + i_imag * d_imag;
               re = l_real;
               im = l_imag;
               tiefe = 0;
               do{
                  xx = re * re;
                  yy = im * im;
                  im = 2.0 * re * im - l_imag;
                  re = xx - yy - l_real;
                  tiefe++;
              }while ((tiefe < rechentiefe) && ((xx + yy) < unendlich));
              if (tiefe < rechentiefe)
```

```
                if (tiefe > zeichentiefe)
                   plot(i_real,i_imag);
                else
               if ((tiefe %2 ) != 0)    /* % = modulo - Operator        */
                      plot(i_real,i_imag);
             i_real++;
             vq_mouse(handle,&klick,&dummy,&dummy);
          }while ((klick == 0) && (i_real < breit));
          i_imag++;
      }while ((klick == 0) && (i_imag < hoch));
      while (klick != 1){
         vq_mouse(handle,&klick,&dummy,&dummy);
      }
   desktop();
}

                              /* ENDE UND ZURUECK ZUM DESKTOP       */
desktop()
{
   v_clsvwk(handle);
   appl_exit();
}

                              /* PLOT POINT                         */
plot(x,y)
int x,y;
{
   pxyarray[0] = x;
   pxyarray[1] = y;
   v_pmarker(handle,1,pxyarray);
}

                              /* EINGABE DER BENUTZERDATEN          */
eingaben()
{
   text(1,1,"Apfelmännchen - Juliamengen - Mandelbrotmenge");
   text(2,1,"=============================================");
   text(4,1,"Rechentiefe ............... [ca. 10..20] ?");
   rechentiefe = maus_input(0,100,4,43);
   text(6,1,"Zeichentiefe .............. [ca.  5..15] ?");
```

```
    zeichentiefe = maus_input(0,100,6,43);
    text(8,1,"Bildbreite ................. [ max. 640] ?");
    breit = maus_input(0,640,8,43);
    text(10,1,"Bildhöhe .................. [ max. 400] ?");
    hoch = maus_input(0,400,10,43);
    text(12,1,"Z-Re-Min   [z. B. -0.75  0.0  1.1  1.6] ?");
    re_min = maus_input(-3,3,12,43);
    text(14,1,"Z-Re-Max   [z. B.  2.25  0.2  1.27 1.8] ?");
    re_max = maus_input(-3,3,14,43);
    text(16,1,"Z-Im-Min   [z. B. -1.2 -1.06 -0.35 -0.05?");
    im_min = maus_input(-2,2,16,43);
    text(18,1,"Z-Im-Max   [z. B.  1.2  -0.9 -0.25 0.05 ?");
    im_max = maus_input(-2,2,18,43);
}


maus_input(w_min,w_max,zeile,spalte)
int w_min, w_max;        /* Min- und Maximalwert */
int zeile, spalte;       /* Input at Position    */
{
int status;              /* Taste unten = 1 sonst 0 */
int mx,skala;
double s_f,zahl;

    if (w_min < 0){
       w_min = w_min * -1;
       w_max = w_max + w_min;
    }
    s_f = w_max / 639.0;  /* Skalierung */
    do{
          vq_mouse(handle,&status,&mx,&dummy);  /* sample mouse */
          zahl = mx * s_f - w_min;
          print(zahl,zeile,spalte);
    }while (status == 0);
    evnt_button(1,1,1,&dummy,&dummy,&dummy,&dummy);
    evnt_button(1,1,0,&dummy,&dummy,&dummy,&dummy);
    return(zahl);
}
```

```
/* Ausgabe des Inhaltes einer Variablen */
/* Uebergabewert an VAR MUSS DOUBLE sein                           */

print(var,zeile,spalte)
int zeile,spalte;
double var;
{
char decimal[80];

   ftoa(var,decimal,6);             /* 6 = Nachkommastellen         */
   decimal[79]=0;                   /* initialisiert als char[80]   */
   v_gtext(handle,width * spalte,height * zeile,decimal);
}

/*                          adressierte                          TEXTAUSGABE
***********************************************/

text(zeile,spalte,string)
int zeile,spalte;
char string[80];
{
   v_gtext(handle,width * spalte,height * zeile,string);
}
```

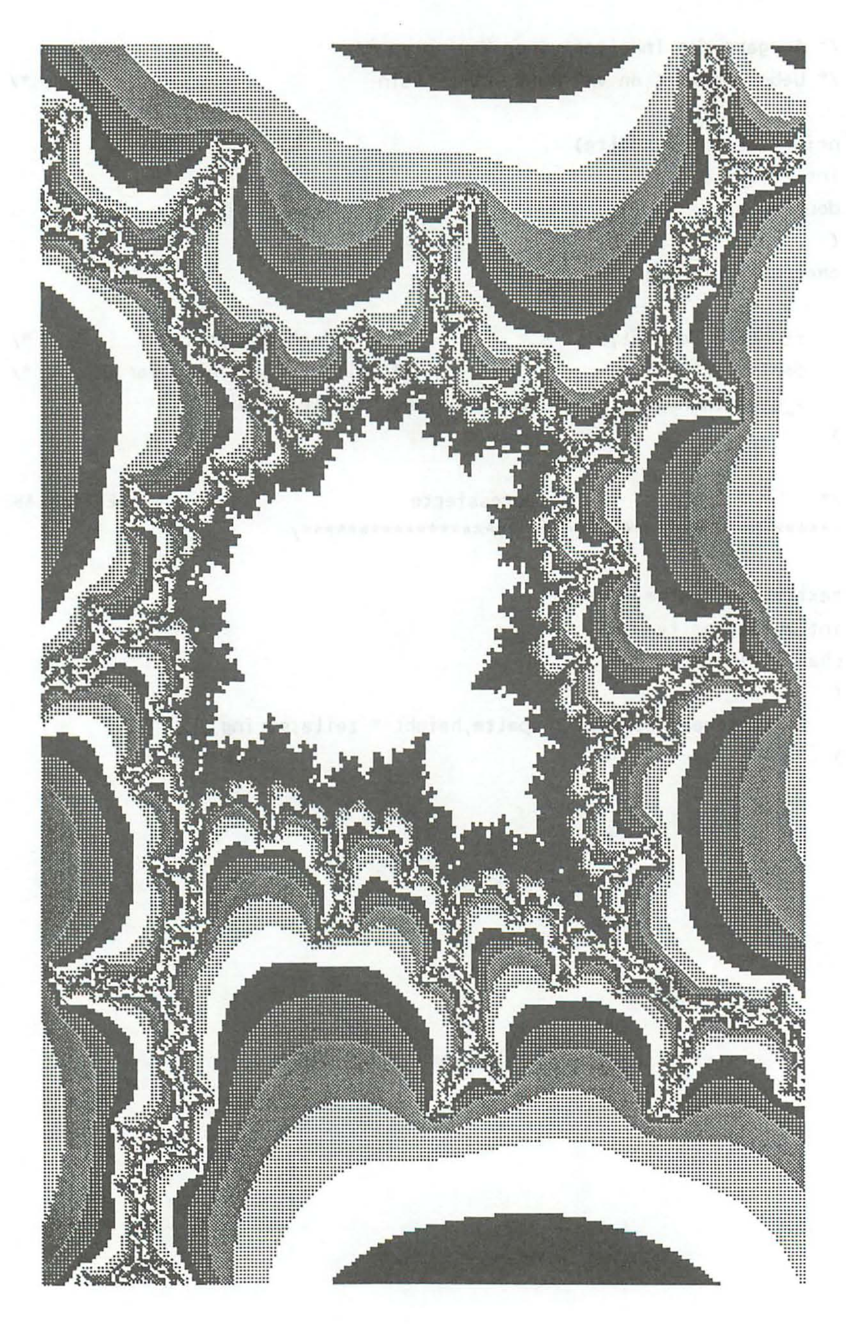


Abb. 53: Apfelmännchen im Bereich 0.14, 0.18, -1.05, -1.02

# 6. Kapitel

# Sound

## 6.1 Der Sound aus dem Computer

Computerspiele ohne Tonuntermalung? - Undenkbar. Zumindest dann, wenn es sich um ein 'professionelles' Spiel handeln soll.

Ping und Plong sind das mindeste, und werden auch bereits seit Jahren realisiert. Zur Synthese solch einfacher Toneffekte reichte es aus, periodisch ein mit einer bestimmten Speicherstelle gekoppeltes Flip/Flop umzuschalten, das mit einem Lautsprecher verbunden war (Apple II). Oder man gab genau ausgeknobelte Bytes in einer noch genauer berechneten Zeitfolge über einen Port aus, an den ein Verstärker angeschlossen war (TRS-80). Damit ließen sich nach entsprechendem Arbeitseinsatz schon ganz ordentliche Töne aus dem Computer entlocken. Doch hörte es sich eben immer noch nach Computer an, konnten schließlich auch nur reine Rechtecksignale erzeugt werden. Außerdem war nur eine einzige Stimme verfügbar.

Doch das Interesse war geweckt. Die Hobbyisten wollten ein besseres Equipment zur Soundsynthese und auch die Profis drängten zur Entwicklung miniaturisierter Synthesizer, die in Orgeln und auch Spielhallenautomaten eingesetzt werden sollten.

Heraus kam dann ein Chip, wie beispielsweise der AY-3-8910 von General Instruments, der sich aufgrund seiner Leistungsstärke durchsetzte. So wurde er bereits kurz nach seiner Entwicklung auf Zusatzkarten für den Apple II und den TRS-80 angeboten, und fand auch einen Steckplatz in den Heim- und Hobbycomputern der nächsten Generation (Colour Genie, Schneider CPC und MSX-Rechner), wo er sich hervorragend bewährte.

Wen wunderts da, daß dieser vielseitige und durch vielfachen Einbau inzwischen auch preisgünstige Chip, beziehungsweise ein YAMAHA YM-2149, einer von vielen Lizenznachbauten, auch im ST Verwendung fand. Läßt er doch mit seinen drei Tongeneratoren, einem zusätzlichen Rausch- wie auch Hüllkurvengenerator und Mischer kaum noch Wünsche offen.

Doch wie entlockt man solch einem Ding die unterschiedlichsten Töne und Geräusche?

Wie veranlaßt man ihn, Instrumente lebensecht zu simulieren? Wie können schließlich ganze Melodien ein- und mehrstimmig gespielt werden?

Und zu guter letzt: Was hat Atari getan, um uns den Gebrauch dieses IC's möglichst komfortabel zu gestalten, d.h., wie läßt er sich von BASIC und Logo aus ansprechen?

All diese Fragen sollen im folgenden geklärt werden, wobei auch auf die wichtigsten Grundlagen zur Tonerzeugung eingegangen werden soll.

## 6.2 Der Ton macht die Musik

Das ist nicht nur eine bloße Redensart, sondern diese Redewendung trifft, wie wir noch sehen werden, gerade im Bereich der Musikerzeugung mittels elektronischer Hilfsmittel mehr zu, als Sie vielleicht glauben.

Wie wohl jeder weiß, 'besteht' ein Ton aus Schwingungen; aus Schwingungen, die durch die Luft übertragen schließlich an unser Ohr gelangen, und dort das Trommelfell in mehr oder weniger schnelle und auch heftige Bewegungen versetzen. Diese Schwingungen, in elekrische Impulse verwandelt, lassen in unserem Gehirn den Eindruck eines Geräusches entstehen. Dabei empfinden wir den Ton umso höher, desto schneller das Trommelfell vibriert.

Doch wie läßt es sich erklären, daß eine Saite, welche in einem Klavier pro Sekunde 440 mal hin und herschwingt, anders klingt, als eine gleich lange Saite und gleich schnell vibrierende Saite in einem Flügel? Ganz klar, wird vielleicht jemand antworten, das eine Instrument hat einen weichen, und das andere einen harten Anschlag.

Doch wie wirkt sich dieser unterschiedliche Anschlag beispielsweise auf den Ton aus ? Um diese Frage beantworten zu können, muß man sich zunächst über den Aufbau eines Tones klar werden.

Unterscheiden lassen sich im wesentlichen die regelmäßigen, periodischen Klänge von den unregelmäßigen, aperiodischen Geräuschen.

Schon die Unterscheidung in Klang auf der einen und Geräusch auf der anderen Seite weist auf die Unterschiede in iher Erscheinungsform hin; als Beispiele lassen sich die von einer Blockflöte erzeugten Töne wie auch die vom Brausen des Windes erzeugten Geräusche nennen.

Doch ist in der Regel kein von einem Instrument erzeugter Ton absolut rein, sondern es handelt sich immer um eine Mischung aus mehreren Tönen, aus Grundschwingung und Oberwellen.

Diese Zusammenhänge hat zu Beginn des 19. Jahrhunderts der französische Mathematiker Joseph de Fourier in seinem Werk 'Theorie analytique de la chaleur' klargestellt.
Er machte sich daran, periodische Funktionen durch Reihen von periodischen Funktionen darzustellen, und mit seinen Theorien schuf er die Voraussetzungen für viele Anwendungen in Mathematik, Physik und Technik.

So entwickelte er das Verfahren der 'harmonischen Analyse von Klängen', bei der eine Schwingung in eine Reihe von reinen Sinusschwingungen und einen konstanten Anteil zerlegt werden.
Er stellte fest, daß die neben der Grundschwingung auftretenden Oberschwingungen immer als ein Vielfaches dieser Grundfrequenz auftreten.

Unter Berücksichtigung dieser Fourierreihenentwicklung ist natürlich auch die Umkehrung des Vorganges, die harmonische Synthese, möglich. Dabei werden die einzelnen Schwingungen addiert, und es ergibt sich eine Resultierende. Auf diese Weise lassen sich andere Schwingungsformen konstruieren, wie zum Beispiel die Rechteck- oder die Sägezahnschwingung.

Neben der Frequenz der Grundschwingung, die die Höhe des Tones festlegt, entscheidet der Oberwellengehalt, die Form der Kurve, über den Klang des Tones.

## 6.3 Die Lautstärkenhüllkurve

Aber nicht nur der Oberwellengehalt allein ist bestimmend für den typischen Klang eines bestimmten Instrumentes. Mindestens ebenso typisch ist das Lautstärkeverhalten, das in Form einer sogenannten Lautstärkehüllkurve für ein jedes Instrument angegeben werden kann.
Dabei unterscheidet man vier verschiedene Phasen: Attack, Decay, Sustain und Release, weshalb die Lautstärkehüllkurve in der Literatur vielfach auch als ADSR bezeichnet wird.

Im einzelnen beschreibt man mit

- Attack den Zeitraum, der vergeht, ehe der Ton beginnend vom ersten Einsatz des Instrumentes an, also von der Lautstärke Null, die maximale Lautstärke erreicht hat;

- Decay die vergangene Zeit, ehe der Lautstärkepegel auf ein konstantes Level zurückgefallen ist, der danach eine gewisse Zeit gehalten wird ehe der Ton verklingt;

- Sustain das zuvor erwähnte konstante Level;

- Release die Zeit, die verstreicht, ehe der Ton vom Sustain-Level wiederum bis zur absoluten Ruhe abgefallen ist.

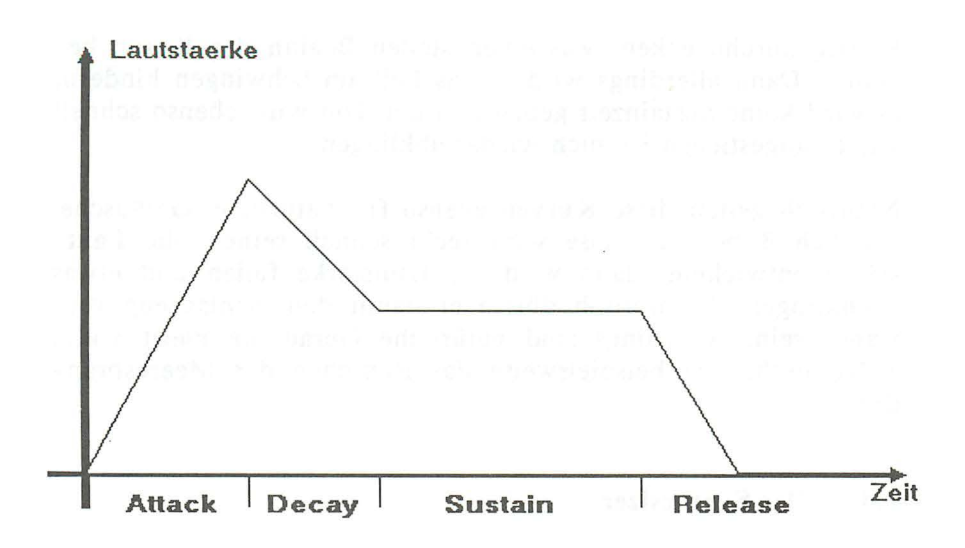



Abb. 54: Beispiel einer Hüllkurve

Stellen Sie sich nun vor, Sie blasen eine Trompete. Dann wird der Ton nicht sogleich beim Öffnen eines Ventiles seine volle Lautstärke haben. Vielmehr wird der Pegel davon abhängig sein, wie stark Sie in das Instrument blasen. Mit steigendem Luftdruck wird die Lautstärke zunächst anschwellen (Attack) und nach kurzer Zeit (Decay) eine gewisse Höhe haben (Sustain), ehe sie danach wieder abfällt (Release), entweder rapide, wenn Sie das Ventil schließen, oder ganz langsam, wenn Ihnen nämlich die Luft ausgeht.

Diese vier Stadien, im Deutschen auch als Anstieg, Abfall, Aushalten und Freigabe bezeichnet, lassen sich bei jedem Instrument beobachten und sorgen für den unterschiedlichen Klang bei gleicher Frequenz.

So wird sich der Lautstärkeverlauf der Pauke eines Schlagzeugs vermutlich als eine ein spitzes Dreieck darstellende Lautstärkenkurve präsentieren, denn der Schlegel wird das gespannte Fell

kräftig durchdrücken, was einen steilen Beginn der Kurve bedeutet. Dann allerdings wird er das Fell am Schwingen hindern, es wird keine Sustainzeit geben und der Ton wird ebenso schnell wie er angestiegen ist auch wieder abklingen.

Natürlich gelten diese Kurven ebenso für natürliche Geräusche. Ein Schuß beispielsweise wird recht schnell seine volle Lautstärke entwickeln, dann wird die Lautstärke fallen und etwas nachklingen. Theoretisch dürfte er damit dem Schlagzeug verwandt sein. Allerdings sind natürliche Geräusche meist völlig aperiodisch, wie beispielsweise das Rauschen der Meeresbrandung.

## 6.4 Der Synthesizer

Dieses Wissen zunutze gemacht hat sich als erster Bob Moog, der im Jahre 1964 seinen Moog-Synthesizer vorstellte.
Er konstruierte damals ein technisches Ungetüm, das es ihm erlaubte, die klangbestimmenden Glieder eines Tones, Frequenz, Oberwellengehalt und Lautstärke, nach Wunsch zu verändern. Selbstverständlich arbeitete dieses Gerät analog, d.h., die verschiedensten Parameter wurden durch unterschiedliche Spannungen realisiert.

Runde zwanzig Jahre später war es dann soweit, daß ein einziges IC, digital gesteuert, zu ähnlichen und teilweise sogar besseren Leistungen fähig ist wie die damaligen Moog-Synthesizer. So besitzt der in den ATARI ST eingesetzte Programmable Sound Chip (PSG) drei Stimmen, was zwar einerseits nur einem dreifingrigen Klavierspieler entspricht, andererseits aber auch schon zu ganz ordentlichen Ergebnissen führen kann.

Hinzu kommt, quasi als vierte Stimme, ein Rauschgenerator, der zur Synthese aperiodischer Geräusche eingesetzt und auch über einen Mischer den Tonkanälen zugespielt werden kann.

Er verfügt über einen programmierbaren Hüllkurvengenerator, der zur Erzeugung der unterschiedlichsten Effekte eingesetzt

werden kann. Selbstverständlich kann auch die Lautstärke eines jeden Tonkanals geregelt werden.

Natürlich entsteht so zunächst wiederum ein Rechtecksignal, schließlich handelt es sich bisher um reine Digitaltechnik. Doch bevor die Signale ausgegeben werden, wandelt ein D/A-Wandler diese in die üblichen Analogsignale um, so daß die Chips der AY-3 Reihe tatasächlich als Synthesizer eingesetzt werden können.

## 6.5 Der Chip

Bei dem in den ATARI ST eingebauten Programmable Sound Generator AY-3-8910 vom General Instruments bzw. dessen Ersatztypen handelt es sich um einen LSI-Chip (Large Scale Integration), der geschaffen wurde, um auch komplexe Töne und Geräusche unter Softwarekontrolle erzeugen zu können.

Insbesondere sieht dieser PSG vor, daß der Prozessor des Computers andere Aufgaben verrichten kann, sobald er den PSG wunschgemäß initialisiert hat. Die AY-3-Soundgeneratoren arbeiten unabhängig von dem sie kontrollierenden Prozessor. Die Arbeitsweise der Chips ist durchweg digital. Alle Steuersignale werden in digitaler Form erwartet. So erklärt sich auch die Tatsache, daß keine weiteren Bausteine benötigt werden und der PSG direkt an einen Microprozessor angeschlossen werden kann.

Diese Verbindung zwischen dem Kontrollprozessor und dem PSG geschieht durch einige Speicherstellen, d.h. bei dem Tongenerator handelt es sich um einen memory-mapped-Baustein. Zur Kommunikation dienen 16 Register, die vom Hauptprozessor, also im Falle des ST vom 68000, beschrieben als auch gelesen werden können. Sämtliche Funktionen des PSGwerden einzig und allein durch diese Register $00 bis $0F kontrolliert. Sobald der Prozessor sie geladen hat, wird der PSG mit seiner Arbeit beginnen. Insbesondere wird er auch so lange einen Ton erklingen lassen, bis er durch entsprechende Einstellung der Register wieder abgeschaltet wird. Der 68000 kann derweil mit der Programmausführung fortfahren.

Wenden wir uns nun dem Aufbau des Soundchips zu. Im wesentlichen besteht dieser aus sechs Funktionsblöcken:

*1. Der Tongenerator*

Dieser produziert, maximal für die drei Kanäle A, B und C, zunächst die gewünschte Frequenz. Dabei handelt es sich immer um eine Rechteckschwingung.

*2. Der Rauschgenerator*

Am Ausgang des Rauschgenerators steht ebenfalls eine Rechteckschwingung zur Verfügung. Deren Frequenz wird allerdings zufällig verändert.

*3. Der Mischer*

Eigentlich sind es sogar drei Mischer (jeweils einer für jeden Kanal). Aufgabe dieses Funktionsblocks ist es, die Ausgangsschwingungen der Tonkanäle mit der Rauschfrequenz zu mischen.

*4. Der D/A-Wandler*

Wie bereits erwähnt, arbeitet der AY-3-PSG durchweg digital, d.h., die Signale müssen zunächst in analoge Form (Spannungsschwankungen) gebracht werden, ehe sie über einen Verstärker an unser Ohr gelangen. Die Ausgangsspannung des Digital-/Analogwandlers liegt dabei zwischen 0 und 1 Volt, abhängig von der eingestellten Lautstärke. Dabei wird auch die logarithmische Empfindlichkeit des menschlichen Ohres berücksichtigt, d.h. die Ausgangsspannung erhöht sich von Stufe zu Stufe nicht um einen konstanten Betrag sondern logarithmisch.

Natürlich muß der D/A-Wandler wissen, welche Lautstärke gewünscht wird. Diese Einstellung erfolgt über vier Bits, d.h. der Regelbereich der Lautstärke sieht Einstellungen von 0 bis 15 vor.

Die zu wandelnden Frequenzen erhält der D/A-Wandler vom Mischer (Ton und/oder Rauschen).

*5. Die Amplitudenkontrolle*

Die Amplitude, also die 'Höhe' einer Schwingung, legt die Lautstärke des Tones fest und wird dem D/A-Wandler von diesem Funktionsblock vorgegeben. Dabei sind grundsätzlich zwei völlig verschiedene Wege gangbar. So kann zum einen eine konstante Lautstärke (im Bereich von 0 bis 15) durch das Laden entsprechender Kontrollregister vorgegeben sein; die Ausgangsamplitude steht dann unter direkter Kontrolle des Hauptprozessors. Zum anderen kann allerdings auch eine variable Lautstärkesteuerung gewählt werden; die Ausgangsamplitude wird dann durch den Hüllkurvengenerator gesteuert.

*6. Der Hüllkurvengenerator*

Dieser erzeugt ein Signal, welches die Amplitudenkontrolle dazu benutzen kann, um eine Amplitudenmodulation des Ausgangssignals der Mischer vorzunehmen.

Soweit betrifft diese Beschreibung der Funktionsblöcke die Soundchips General Instruments AY-3-8910, AY-3-8912, AY-3-8913 und natürlich auch Yamaha YM-2149. Doch bei den Chips 2149 und 8910 lassen sich zwei zusätzliche Funktionsblöcke entdecken, die absolut nichts mit der Tonerzeugung zu tun haben. Es handelt sich dabei um zwei 8-Bit-I/O-Ports, die an dieser Stelle erwähnt werden, weil sie innerhalb des ATARI ST auch genutzt und somit für uns noch wichtig werden. Der AY-3-8912 unterscheidet sich von diesen

beiden Typen dadurch, daß er nur über einen Port verfügt, der AY-3-8913 schließlich ist mit keinem Port ausgestattet.

Das Zusammenspiel sämtlicher Funktionsblöcke des PSG macht noch einmal die folgende Abbildung deutlich.

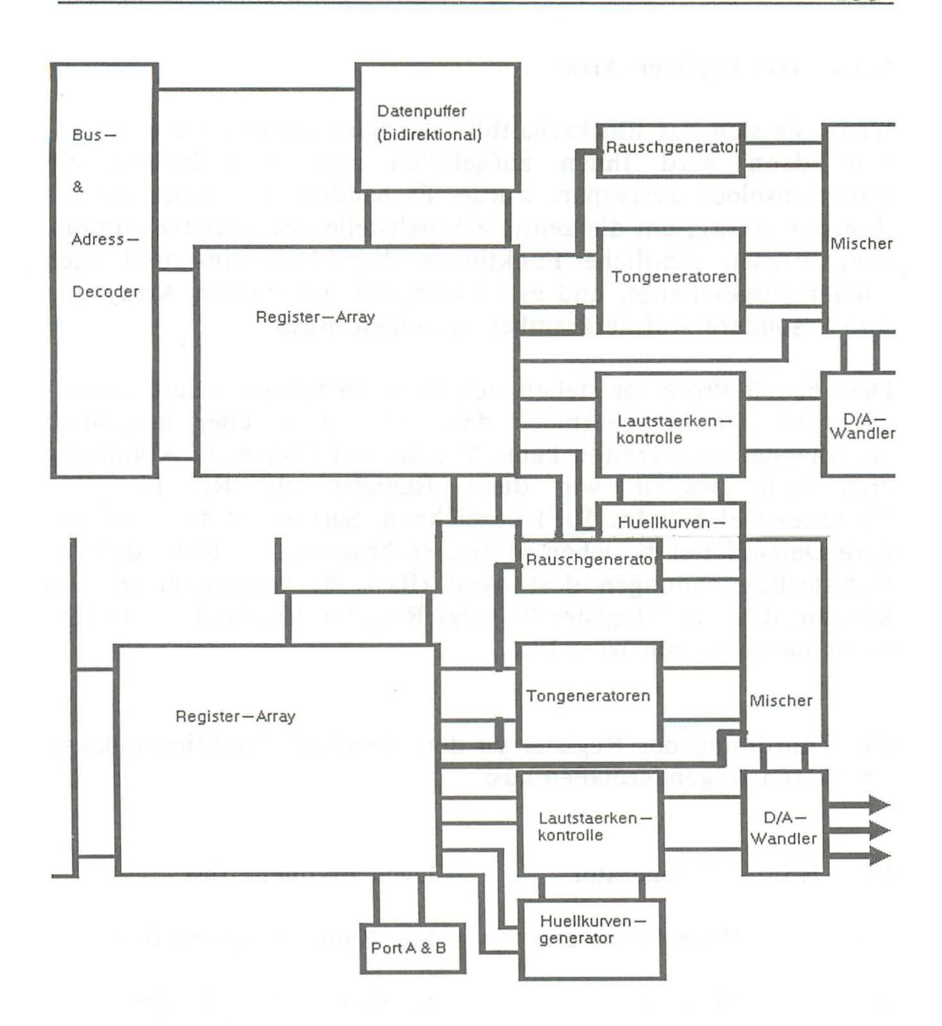



Abb. 55: Blockschaltbild des PSG

### 6.5.1 Das Register-Array

Wenn Sie sich das Blockschaltbild des Soundchips angesehen haben, dann wird Ihnen aufgefallen sein, daß bislang ein Funktionsblock ausgespart wurde. Es handelt sich dabei um das Register-Array, um die zentrale Schaltstelle des Soundgenerators. Hier werden sämtliche Funktionen des Chips ein- und auch wieder ausgeschaltet, und es ist klar, das wir diesem Array unsere besondere Aufmerksamkeit schenken müssen.

Dem 68000-Prozessor stellen sich diese 16 Schreib- und Leseregister als ein Speicherblock dar, auf die er über bestimmte Speicherstellen zugreifen kann. Wie aus der folgenden Abbildung ersichtlich, wollen wir diese Register als R0 bis R15 (Hexadezimal $00 bis $0F) bezeichnen. Sofern Sie sich auf andere Datenblätter beziehen sollten, so beachten Sie bitte, daß die Registerbezeichnungen dort oktal (Basis 8) vorgenommen sein können, d.h., auf Register 07 folgt Register 10 (das höchste Register hat dann den Wert 17).

Die Zuordnung der Register zu den einzelnen Funktionsblöcken sieht dann folgendermaßen aus:

| | | |
|---|---|---|
| R0 - R5 | Tongenerator | legen Frequenz fest |
| R6 | Rauschgenerator | legt Rauschfrequenz fest |
| R7 | Mischer | schaltet Ton und/oder Rauschen auf den Kanälen A, B und C ein |
| R8 - R10 | Amplitudenkontrolle | Lautstärkeregelung für A, B, und C |
| R11 - R13 | Hüllkurve | legen Frequenz und Aussehen der Hüllkurve fest |
| R14 - R15 | Port A und B | im ST für Floppy I/O genutzt |

| REG | FUNKTION | BEREICH | BIT 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| 0<br>1 | Frequenz Kanal A | 0 – 4095 | 8–Bit–Feineinstellung<br>x x x x Grob |
| 2<br>3 | Frequenz Kanal B | 0 – 4095 | 8–Bit–Feineinstellung<br>x x x x Grob |
| 4<br>5 | Frequenz Kanal C | 0 – 4095 | 8–Bit–Feineinstellung<br>x x x x Grob |
| 6 | Rauschperiode | 0 – 31 | x x x 5 bit Periode |
| 7 | Enable | | Port Rauschen Ton<br>B A C B A C B A |
| 8<br>9<br>10 | Lautstaerke A<br>Lautstaerke B<br>Lautstaerke C | 0 – 15 oder H<br>0 – 15 oder H<br>0 – 15 oder H | x x x H 1 1 1 1<br>x x x H 1 1 1 1<br>x x x H 1 1 1 1 |
| 11<br>12 | Huellkurven–<br>periode | 16 bits =<br>0 – 65535 | 8 bit Feineinstellung<br>8 bit Grobeinstellung |
| 13 | Huellkurve | | x x x x h h h h |
| 14<br>15 | I/O Port A<br>I/O Port B | | 8–Bit–Parallelport<br>8–Bit–Parallelport |

Abb. 56: Die Register des Soundchips

### 6.5.1.1 Die Register im einzelnen

*1. Die Kontrolle des Tongenerators*

Zur Festlegung der Tonfrequenzen für die drei Kanäle A, B und C stehen die Register R0, R1, R2, R3, R4 und R5 zur Verfügung, für jeden Kanal somit zwei (8-Bit-) Register. Damit stehen jedem Kanal 16 Bit zur Verfügung, von denen allerdings nur 12 Bit genutzt werden. Womit ein Regelbereich von 0 bis 4095 abgedeckt wird. Die Rechteckschwingungen der drei Tongeneratoren werden dann folgendermaßen erzeugt:

Das am PSG anliegende Clock-Signal wird zunächst durch 16 dividiert, anschließend erfolgt eine weitere Division durch den 12-Bit-Wert der betreffenden Register. Diese sind unterteilt in Register zur Grob- (R1, R3, und R5, auch als Coarse Tune Register bezeichnet, je 4 Bit, MSB des 12-Bit-Wortes) und Feinabstimmung (R0, R2 und R4, Fine Tune, 8 Bit, LSB).

*2. Die Kontrolle des Rauschgenerators*

Entsprechend wird auch die Einstellung des Rauschgenerators vorgenommen. Von den 8 Bits in Register R6 werden die fünf Bits 0 bis 4 genutzt, d.h., der Einstellbereich des Rauschgenerators erstreckt sich über einen Bereich von 0 bis 31. Auch hier wird zunächst das Clock-Signal durch sechzehn und dann durch den Wert in R6 dividiert.

*3. Die Kontrolle des Mischers*

R7 stellt sich als Multifunktionsregister dar, um die verschiedenen Funktionsblöcke (auch die beiden Ports!) ein- bzw. auszuschalten. Dabei bedeutet ein gesetztes Bit, daß die entspechende Funktion ausgeschaltet ist!

Die niederwertigsten Bits (0 - 2) kontrollieren die Tongeneratoren A, B, und C. Findet der PSG nur hier Nullen, so erscheint am Ausgang die reine Rechteckfrequenz, die mit R0 bis R5 eingestellt worden ist. Die Bits 3 bis 5 mischen den entsprechenden

Kanälen Rauschen zu, sollten nur diese gelöscht sein, erscheint die pure Rauschfrequenz am Ausgang.

B6 und B7 schließlich sind für die I/O-Ports (Register 14 und 15) zuständig. Wichtig im ST ist hier besonders Bit 7, denn dieses kontrolliert Port B der für den Diskettenzugriff zuständig ist. Aus diesem Grunde sollten wir vor jedem Zugriff auf den Soundchip zunächst dieses Register auslesen, den Wert zwischenspeichern und später wieder restaurieren (Jetzt wissen Sie auch, warum bei der Tonerzeugung unter Logo die LED's der Laufwerke aufleuchten - scheint hier doch noch ein Fehler zu stecken).

*4. Regelung der Lautstärke*

R8, R9 und R10 steuern das Ausgangssignal der D/A-Wandler und somit die Lautstärke der Kanäle A, B und C.
Bit 5 bis 7 werden nicht genutzt, Bit 4 legt fest, ob die Regelung über eine Hüllkurve erfolgen soll (dann muß B4 gesetzt sein) oder ob der Ton mit konstanter Lautstärke gespielt wird. Ist B4 = 0, legt der Wert der Bits 0 bis 3 die Signalstärke fest, was einem Regelbereich von 0 bis 15 entspricht.

*5. Die Steuerung des Hüllkurvengenerators*

Die für Soundoperationen verbleibenden drei Register R11, R12 und R13 dienen der Einstellung der verschiedensten Hüllkurven. Um dabei möglichst vielfältige Variationsmöglichkeiten zur Verfügung zu haben, wurden zwei voneinander unabhängige Methoden im PSG implementiert.

So sieht die erste Methode eine Variation der Frequenz der Hüllkurven über die Register R13 und R14 vor. Sie werden zusammengefaßt zu einem 16-Bit-Wort, was einen Einstellbereich von 0 bis 65535 erlaubt. Wie gehabt werden die Register wiederum unterteilt in Envelope Coarse Tune (R12) und Envelope Fine Tune (R11). Um dann die Hüllkurvenfrequenz festzulegen, dividiert der PSG zunächst die Clock-Frequenz durch 256 und das Ergebnis anschließend durch den programmierten 16-Bit-Wert.

Das zweite Verfahren hingegen sieht vor, unterschiedliche ADSR-Hüllkurven (Envelope Shape/Cycle) auszuwählen. Dazu dienen die untersten vier Bits des Register R13, die mit Hold, Alternate, Attack und Continue bezeichnet werden (die Bits 4 bis 7 werden nicht benutzt). Die Arbeitsweise des PSG sieht nun vor, daß er die feststehende Hüllkurvenfrequenz noch einmal durch 16 dividiert und somit einen Zyklus in 16 einzelne Schritte einteilt. Je nach angewähltem Hüllkurvenmuster wird ein Zähler mit diesen Schritten aufwärts oder abwärts gezählt, mit den Werten dieses Zählers wird die Amplitudenkontrolle angesteuert.

Dabei gibt es sich wiederholende Muster oder aber auch Hüllkurven, die jeweils nur einen einzigen Zyklus betreffen. Einen Überblick über die zur Verfügung stehenden Hüllkurven gibt die folgende Abbildung.

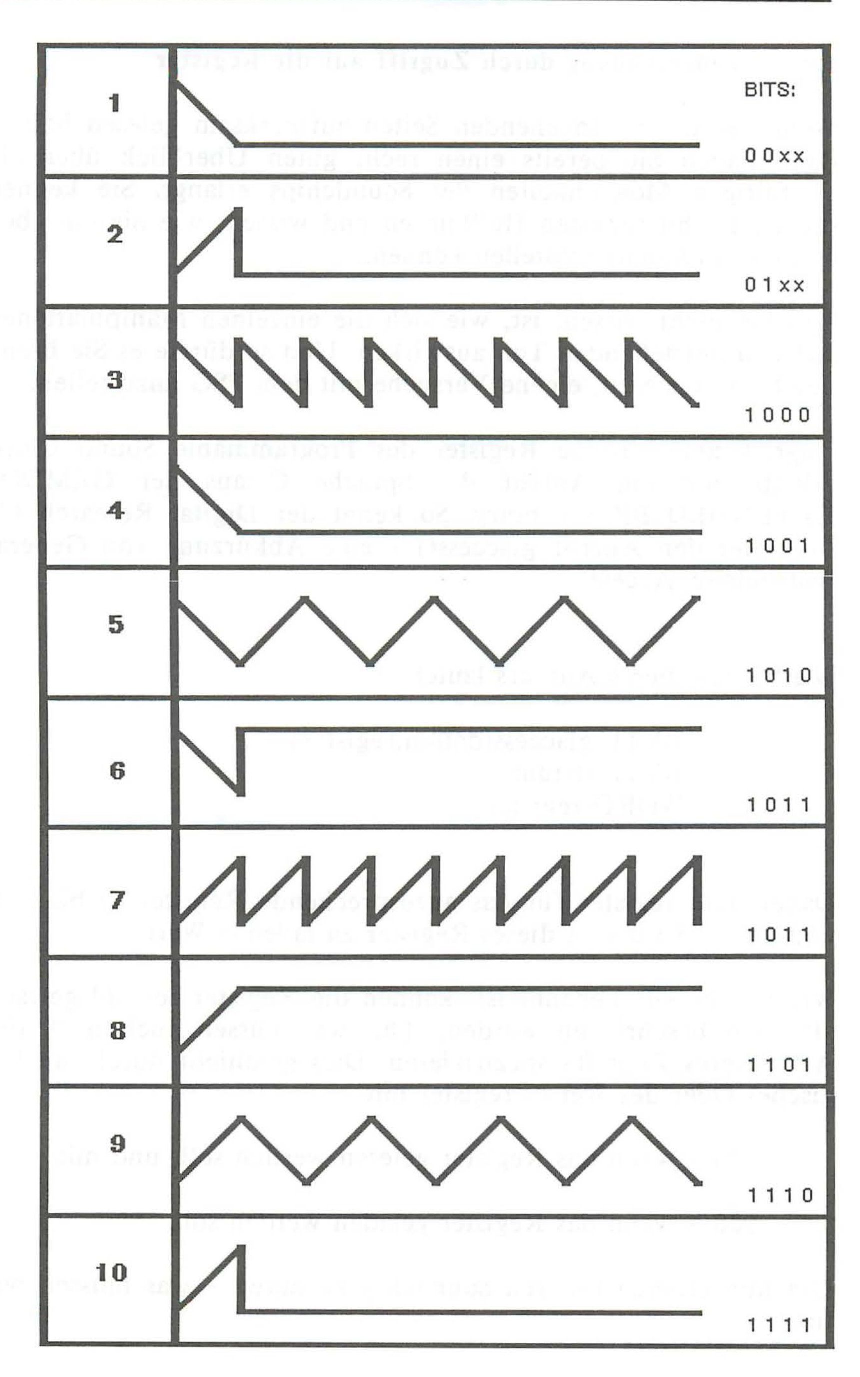



Abb. 57: Die Hüllkurven des PSG

## 6.6 Tonerzeugung durch Zugriff auf die Register

Wenn Sie die vorangehenden Seiten aufmerksam gelesen haben, dann haben Sie bereits einen recht guten Überblick über die vielfältigen Möglichkeiten des Soundchips erlangt. Sie kennen die unterschiedlichsten Hüllkurven und wissen, wie Sie eine bestimmte Frequenz einstellen können.

Was Sie nicht wissen, ist, wie sich die einzelnen Manipulationen auf den entstehenden Ton auswirken. Und so dürfte es Sie brennend interessieren, eigene Versuche mit dem PSG anzustellen.

Zugriff auf einzelne Register des Programmable Sound Chips erlaubt uns ein Aufruf der Sprache C aus der GEMDOS EXTENDED BIOS Library. So kennt der Digital Research C-Compiler den Aufruf giaccess() - eine Abkürzung von General Instruments Access.

Die Syntax dieses Aufrufs lautet:

```
BYTE giaccess(datum,register)
BYTE datum;
WORD register;
```

Dabei steht register für das anzusprechende Register (0 bis 15), und datum ist der in dieses Register zu ladende Wert.

Wie uns bereits bekannt ist, können die Register sowohl gelesen als auch beschrieben werden, d.h., wir müssen auch noch die Art unseres Zugriffs spezifizieren. Dies geschieht durch ein logisches Oder des Wertes register mit

$00 - wenn das Register gelesen werden soll, und mit

$80 - wenn das Register geladen werden soll.

Um nun endlich unseren Soundchip zu hören - was müssen wir tun?

Insgesamt werden es wohl mindestens drei Schritte sein, nämlich:

1. die Tonfrequenz festlegen (R0 bis R5)
2. den gewünschten Kanal aktivieren (R7)
3. die Lautstärke auf einen Wert ungleich Null einstellen (R8 bis R10)

Bevor wir uns im folgenden genauer mit den Werten befassen, wollen wir doch schnell einen Versuch wagen. Anweisungen für den PSG könnten beispielsweise lauten:

zu 1:

```
giacces(0,0 + 128);
giaccess(100,1 + 128);
```

zu 2:

```
giaccess(...,7 + 128);
```

zu 3:

```
giaccess(15,8 + 128);
```

Auffällig ist hier die 128, die zu jedem Registerwert addiert wird. Doch handelt es sich hierbei um die Maske, die besagt, daß in die Register geschrieben werden soll, anstatt sie zu lesen. Erinnern wir uns nun noch, daß wir den Status des Port-Registers 15 (festgelegt mit dem Wert in R7) retten wollen, so daß wir ihn später restaurieren können, schließlich wollen wir den Kontakt zu unserer Floppystation nicht verlieren. Und schon können wir folgendes kurze C-Programm schreiben, welches einen Ton auf Kanal A mit maximaler Lautstärke erzeugt:

```
                                 /* include files                        */
#include "obdefs.h"              /* erst mal alle - denn wer weiP, was   */
#include "define.h"
#include "gemdefs.h"
#include "gembind.h"
#include "osbind.h"

                                 /*globale variablen                     */

int contrl[12];
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];                 /* genug Platz fuer alle Faelle         */
int handle,i;                    /* virtual workstation handle           */
int width, height;
int pxyarray[12];                /* Array fuer x,y Koordinaten           */
int pxy[4];
int int_in[11];                  /* Eingabe ins GSX Array                */
int int_out[57];                 /* Ausgabe vom GSX Array                */
int dummy;

int ports;                       /* Status der PSG Register 14/15        */


main()
{
    appl_init();                  /* Arbeitsstation initialisieren       */
    handle=graf_handle(&width,&height,&dummy,&dummy);
    open_vwork(intin,&handle,intout);
    sound();                      /* Tonroutine ausfuehren               */
    click();                      /* und auf Maustastendruck warten      */
    Giaccess(0,8 + 128);          /* dann: Lautstärke von A auf 0        */
    Giaccess(ports, 7 + 128);     /* und: Portzustand wiederherstellen   */
    v_clsvwk();                   /* Arbeitsstation schliessen           */
    appl_exit();                  /*  und zum Desktop                    */
}
```

```
open_vwork()
{
int i;
      for (i = 1; i <10; i++){
        int_in[i] = 1;           /* init int_in array: linetype, farbe, */
        }                        /* fillstyles usw.                     */
      int_in[10] = 2;            /* benutze RC - Koordinaten            */
      v_opnvwk(int_in, &handle, int_out); /* jetzt kanns losgehen ... */
}


click()                          /* wartet auf Mausklick (links)        */
{
      evnt_button(1,1,1,&dummy,&dummy,&dummy,&dummy);
}


sound()
{
   ports = Giaccess(dummy,7);    /* Portstatus auslesen                 */
   Giaccess(0x8E,0 + 128);       /* Frequenz =
   Giaccess(0,1 + 128);
   Giaccess(0x3E,7 + 128);       /* Tonerzeugung nur mit Kanal A        */
   Giaccess(15,8 + 128);         /* volle Lautstaerke                   */
}
```

Nach dem Start der compilierten Fassung sollte ein recht lauter, reiner Ton aus dem Monitor ertönen. Überlegen wir uns nun, welchen Ton wir da eigentlich erzeugt haben.

Wie zuvor erwähnt, erzeugt der Sound-Generator sämtliche Töne durch Frequenzteilung aus der von außen angelegten Master-Clockfrequenz, mit der er gespeist wird. Im Falle des Atari ST handelt es sich dabei um einen Takt von 2 Mhz. Intern wird dieser Takt zunächst durch 16 geteilt, was zu einer Steuerfrequenz von 125000 Hertz führt. Ein weiterer Teiler, welcher durch die Werte in den Registern R0 und R1 (für Kanal A) programmiert ist, erzeugt schließlich die Ausgangsfrequenz.

Nach der Gleichung

$$\text{Tonperiode} = \frac{\text{Taktfrequenz}}{\text{Frequenz} * 16}$$

kann die Tonperiode zu einer gegebenen Taktfrequenz (hier die 2000000 Hertz) berechnet werden.

Durch Umstellung gelangen wir zu der Gleichung:

$$\text{Frequenz} = \frac{\text{Taktfrequenz}}{\text{Periode} * 16}$$

Wir hatten R0 mit $1C und R1 mit $01 geladen, diese Hexadezimalzahl entspricht dem Wert 284. Setzen wir die Werte in obige Gleichung ein, erhalten wir ungefähr den Wert 440 Hz. Somit haben wir den Kammerton A erzeugt.

Es ist uns nun möglich, jeden beliebigen Ton zu erzeugen. Wirklich jeden? Nun, überlegen wir weiter. Die Periode unserer Tongeneratoren A, B und C werden durch einen 12-Bit-Wert festgelegt; Grenzwerte sind somit 0 und 4095. Die Grenzfrequenzen unseres Systems betragen demnach 125000 Hz (Periode 0) und 30.53 Hz, der Frequenzgang unseres Computers erstreckt sich somit von einem schon recht ordentlichen Baß bis hin zu - für uns schon wieder unhörbaren - höchsten Tönen. Allerdings muß an dieser Stelle angemerkt werden, daß wir kaum eine Frequenz ganz exakt einstellen können, sondern immer mit einer geringen Verschiebung rechnen müssen. Die Ursache findet sich darin, daß wir unseren PSG schließlich immer nur mit Integer-Werten laden können, die Nachkommastellen gehen unweigerlich verloren. (Beachten Sie dazu die Tabelle im Anhang!)

An dieser Stelle will ich Ihnen nun ein BASIC-Programm vorstellen, das alle notwendigen Umrechnungen vornimmt. Als Ergebnis erhalten Sie für eine gegebene Frequenz die beiden Hex-

zahlen, die Sie in die Register des Tongenerators einschreiben müssen.

```
10    ' CHANGE.BAS : wandelt Dezimalzahl im Bereich von
20    '              0 bis 32767 (16-bit) in Hexzahl um
30    '
40    clearw 2 : fullw 2
50    dim digit$(16)
60    data 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
70    for digit=0 to 15 : read digit$(digit) : next
80    input"Dezimalzahl: ";dezimal
90    if dezimal < 0 then goto 80
100   if dezimal<=255 then hibyte=0 : lowbyte=dezimal: goto 130
110   lowbyte = dezimal mod 256
120   hibyte = (dezimal - lowbyte) / 256
130   dez = hibyte : gosub change : dez = lowbyte : gosub change
140   print
150   goto 80
160   '
170   rem dezimalzahl zwischen 0 und 255 in hex wandeln
180   change:
190   z1=int(dez/16)
200   z2=dez and 15
210   hexzahl$=digit$(z1) + digit$(z2)
220   print hexzahl$;
230   return
```

Zum Schluß dieses Abschnittes noch ein Wort zu unserem C-Listing. Wenn Sie einen jeden Giaccess-Aufruf analysieren wollen, so wird Ihnen das nur gelingen, wenn Sie entweder schon über sehr viel Praxis verfügen und die Registernummern alle im Kopf haben oder aber, wenn Sie stets die Abb. 56 aufschlagen.

Im folgenden wollen wir uns daher ebenfalls nicht der Postulierung eines selbsterklärenden Programmierstils verschließen und als ersten Schritt zu besser lesbaren Programmen extensiven Gebrauch von symbolischen Konstanten machen.

Das bedeutet, wir definieren leicht zu merkende Worte wie KANAL_A und VOLUME_A, die wir von nun an auch in sämtlichen Sound-Programmen verwenden wollen. Um jedoch die Listings nicht zu sehr aufzublähen, schreiben wir alle diese Definitionen in eine Datei SOUND.H, die wir - wie bereits von den Definitionsfiles zu GEM her bekannt - mittels #include in unsere Programme einbinden wollen.

Aussehen soll diese Include-Datei folgendermaßen:

```
#define KANALA_FEIN 0
#define KANALA_GROB 1
#define KANALB_FEIN 2
#define KANALB_GROB 3
#define KANALC_FEIN 4
#define KANALC_GROB 5
#define RAUSCHKANAL 6
#define KANALWAHL 7
#define PORT_STATUS 7
#define VOLUME_A 8
#define VOLUME_B 9
#define VOLUME_C 10
#define ENV_FINE 11
#define ENV_GROB 12
#define ENV_SHAPE 13
#define WRITE 128

#define A_EIN 0x3E          /* Tonerzeugung          */
#define B_EIN
#define C_EIN
#define AB_EIN
#define AC_EIN
#define BC_EIN
#define ABC_EIN
#define RA_EIN              /* Rauschen              */
#define RB_EIN
#define RC_EIN
#define RAB_EIN
#define RAC_EIN
```

```
#define RABC_EIN

#define A_AUS          /* Tonerzeugung        */
#define B_AUS
#define C_AUS
#define AB_AUS
#define AC_AUS
#define BC_AUS
#define ABC_AUS

#define RA_AUS         /* Rauschen            */
#define RB_AUS
#define RC_AUS
#define RAB_AUS
#define RAC_AUS
#define RBC_AUS
#define RABC_AUS
```

Unter Benutzung dieser Definitionen würde sound() dann folgendermaßen aussehen:

```
ports = Giaccess(dummy,PORT_STATUS);
Giaccess(0x1C,KANALA_FEIN + WRITE)
Giaccess(0x01,KANALA_GROB + WRITE)
Giaccess(A_EIN,KANALWAHL + WRITE)
Giaccess(15,VOLUME_A + WRITE)
```

Zwar mehr Tipparbeit, aber doch wohl besser lesbar, oder?

## 6.7 Eine sinnvolle Anwendung

Wer nun statt weiter zu forschen lieber ein Programm erstellen möchte, dem auch ein praktischer Nutzen zugesprochen werden kann, der sollte sich mit dem folgendem Listing eines Tongenerators beschäftigen. Dieser soll, wie auch entsprechende im Handel erhältliche Analoggeräte, ein möglichst breites Frequenzband stufenlos abdecken können.

Da sich an unserem Computer nun aber nirgends ein Drehknopf befindet und ein Eintippen der benötigten Frequenzen kaum schnell genug möglich ist, wollen wir die Maus zu diesem Zweck entfremden. Das heißt, wir wollen auf dem Schirm einen horizontalen Balken programmieren, dessen linker Rand der Frequenz 30.5 Hz und dessen rechter Rand einem Maximalwert entsprechen soll. Um dann eine genaue Aussage über die gerade eingestellte Frequenz machen zu können - schließlich werden wir per Augenmaß kaum den richtigen Wert herausfinden - soll diese darunter digital angezeigt werden.

Im wesentlichen stellen sich uns damit vier Aufgaben:

1. Die Abfrage der Mausposition
2. Die Umrechnung der x-Koordinate in die Periode
3. Das Laden der Sound-Register
4. Die Anzeige eines Variableninhaltes auf dem Bildschirm

Nun, Punkt 3 dürfte uns keine Probleme bereiten. Auch Punkt 1 ist recht unproblematisch, stellt uns die GEM-Bibliothek doch den Aufruf evnt_mouse zur Verfügung, welcher bei einem Mausclick unter anderem die genaue x- und y-Position zurückgibt. Mit diesen Werten und unseren Erfahrungen zum Thema Skalierung aus dem Grafikteil läßt sich Aufgabe 2 - unter Zuhilfenahme zuvor angeführter Formel - erledigen.

Das größte Problem, zumindest für jemand, der mit C gerade erst angefangen hat, dürfte es sein, den Zahlenwert Periode auf dem Bildschirm darzustellen. Schließlich gibt es keine Print-Anweisung der gewohnten Form, d.h. es gibt sie mit printf() schon, doch ist der Digital Research C-Compiler im Gegensatz zum GST-C nicht in der Lage, die Standardausgabefunktionen, zu der die Funktion printf() gehört, gemeinsam mit den Gem-Routinen zu benutzen. Und das Programm möglichst auf allen C-Compilern laufen soll - schließlich will ich Ihnen nicht vorschreiben, welchen Sie sich zu kaufen haben -, muß auf printf() verzichtet werden.

Doch kennen wir bereits aus unserem Grafikteil v_gtext(), eine Funktion zur Ausgabe von Strings an einer Grafikcursorposition. Alles was wir zu tun haben, dürfte demnach sein, unsere Integerzahl Periode in einen String zu verwandeln. Dazu benutzen wir die C-funktion ftoa(a,b,c) - Float to ASCII. Die notwendigen Parameter sind

a - die zu wandelnde Zahl,
b - der Buffer vom Typ char, der die Zeichen aufnehmen soll,
c - die Anzahl der Nachkommastellen.

Unter Benutzung dieses Aufrufs schreiben wir eine Funktion printvar(x), die genau das tut, was ihr Name verspricht.

Außerdem kommen natürlich wieder die üblichen Ergänzungen hinzu, wie Öffnen der Workstation, Abfrage auf Programmende usw. Insgesamt werden wir folgendes Programm formulieren: Nach erfolgreicher Compilierung sind wir dann beispielsweise in der Lage, selbst einmal den Frequenzgang unseres Cassettenrekorders zu überprüfen und zu testen, welche Cassette auf unserem Gerät denn nun die besten Ergebnisse bringt!

## 6.8 Soundmanipulationen durch die Hüllkurve

Soviel zu den reinen Tönen. Ganz interessant wird die Sache, wenn wir in der Lage sind, die erzeugten Rechteckschwingungen so zu manipulieren, daß wir in der Natur auftretende Klänge und Geräusche nachahmen können.

Schreiben wir ein Schießspiel, so soll jede Betätigung des Feuerknopfes auch ein Schußgeräusch anstelle eines Piepsers ertönen lassen. Programmieren wir eine Melodie, dann wäre es schön, wenn unser ST sich dann beispielsweise wie ein Klavier und nicht wie ein Computer anhören würde.

Um diesem Ziel ein wenig näher zu kommen, wollen wir uns nun ein wenig mit den Hüllkurven befassen. Betrachten wir dazu noch einmal die Abb. 57, welche die zur Verfügung ste-

henden zehn Hüllkurven übersichtlich darstellt. Sie alle stellen grafisch dar, wie der PSG die Lautstärke des erzeugten Tones von einer Anfangslautstärke bis zu einer Endlautstärke regelt. Allerdings fällt auf, daß sich bei einigen Einstellungen das Kurvenmuster immer wiederholt (3,5,7 und 9), von einem Endlevel im strengsten Sinn kann also eigentlich gar nicht gesprochen werden. Diese Wellenformen sind so zu verstehen, daß das gewählte Muster ständig wiederholt wird.

So läßt sich die Kurve 9 folgendermaßen interpretieren:

Bei Beginn der Aktion stellt der PSG eine Lautstärke von 0 ein - wir hören also gar nichts. Anschließend wird die Lautstärke stetig bis zur Maximalstärke 15 erhöht, - was dem ansteigenden Teil der Kurve entspricht. Der sich daran anschließende absteigende Teil bedeutet entsprechend eine langsame Senkung der Lautstärke, bis schließlich wiederum nichts mehr zu hören ist und der ganze Prozeß von vorn beginnen kann.

Der Effekt einer solchen Kurve dürfte sich demnach beispielsweise für die Simulation eines sich annähernden und wieder entfernenden Zuges oder etwas ähnlichem eignen.

Die Richtigkeit dieser Überlegungen beweisen wir, indem wir die entsprechende Kurvenwahl in unser Testprogramm einbauen:

```
/***************************************************************************/
/* sub2.c - Programmierung einer Huellkurve                              */
/***************************************************************************/
sound()
{
   ports = Giaccess(dummy,PORT_STATUS);/* Portstatus auslesen          */
   Giaccess(0x1c,KANALA_FEIN + WRITE); /* Frequenz = 440 Hz             */
   Giaccess(0x01,KANALA_GROB + WRITE);
   Giaccess(A_EIN,KANALWAHL + WRITE);  /* Tonerzeugung nur mit Kanal A*/
   Giaccess(ENV_VOLUME,VOLUME_A + WRITE);/* Hüllkurve benutzen         */
   Giaccess(ENV_9,ENV_SHAPE + WRITE);  /* Huellkurve 9 aktivieren       */
   click();                            /* auf Maustastendruck warten    */
```

```
   Giaccess(ALLES_AUS,KANALWAHL+WRITE);/* und PSG abschalten          */
   Giaccess(ports,PORT_STATUS + WRITE);/* Portzustand wiederherstellen*/
}
```

Die Benutzung unseres Definitionsfiles und die Erläuterungen im Programmtext dürften weitere Erklärungen überflüssig machen, so daß wir uns gleich dem nächsten Aspekt der Tongestaltung zuwenden können.

In der Registerübersicht haben Sie sicher auch den mit EP bezeichneten Abschnitt entdeckt, zu dem bislang noch gar nichts gesagt wurde. EP steht für Envelope Period, also für die Periode der Hüllkurve. Damit, in der Praxis vielmehr mit den Werten in den Registern 11 und 12, geben wir dem Soundchip an, wie lange er sich für das Ansteigen der Lautstärke Zeit lassen soll. Bekanntlich werden die Register dabei zu einem 16-Bit-Wert zusammengefaßt, der Regelbereich sieht demnach Perioden von 1 bis 65535 vor. Oder, in Hertz ausgedrückt, Frequenzen von 125000 bis ungefähr 1.9 Hz. Wobei die hohen Schwingungen unserem Ohr dann wieder als eigener Ton erscheinen werden. Überzeugen Sie sich davon, indem Sie im folgenden Beispiel 0xFF gegen 0x00 austauschen:

```
/***********************************************************************/
/* sub3.c - Manipulation der Huellkurvenfrequenz                       */
/***********************************************************************/
sound()
{
   ports = Giaccess(dummy,PORT_STATUS);/* Portstatus auslesen          */
   Giaccess(0x1c,KANALA_FEIN + WRITE); /* Frequenz = 440 Hz            */
   Giaccess(0x01,KANALA_GROB + WRITE);
   Giaccess(A_EIN,KANALWAHL + WRITE);  /* Tonerzeugung nur mit Kanal A*/
   Giaccess(ENV_VOLUME,VOLUME_A + WRITE);
   Giaccess(ENV_9,ENV_SHAPE + WRITE);  /* Huellkurve 9 aktivieren      */
   Giaccess(0xFF,ENV_FEIN + WRITE);    /* Huellkurvenfrequenz:         */
   Giaccess(0xFF,ENV_GROB + WRITE);    /* Periode = 65535              */
   click();
```

```
  Giaccess(ALLES_AUS,KANALWAHL+WRITE);/* und PSG abschalten           */
  Giaccess(ports,PORT_STATUS + WRITE);
}
```

Durch geschickte Wahl der Hüllkurve als auch der Hüllkurvenfrequenz lassen sich nun die meisten Klangfarben der unterschiedlichsten Musikinstrumente und Geräusche nachahmen. Beachten müssen Sie dazu nur, daß es sich in der Natur meist um aperiodische Laute handelt - weshalb Sie den Rauschgenerator anstelle der Tongeneratoren benutzen. Für diesen gilt natürlich ebenfalls das zuvor gesagte.

```
/*************************************************************************/
/* sub4.c - Meeresrauschen                                               */
/*************************************************************************/
sound()
{
  ports = Giaccess(dummy,PORT_STATUS);/* Portstatus auslesen          */
  Giaccess(0,0 + WRITE);
  Giaccess(0,1 + WRITE);
  Giaccess(0,2 + WRITE);
  Giaccess(0,4 + WRITE);
  Giaccess(0,5 + WRITE);
  Giaccess(31,6 + WRITE);
  Giaccess(199,7 + WRITE);
  Giaccess(16,8 + WRITE);
  Giaccess(16,9 + WRITE);
  Giaccess(16,10 + WRITE);
  Giaccess(0,11 + WRITE);
  Giaccess(32,12 + WRITE);
  Giaccess(14,13 + WRITE);
  click();
  Giaccess(ALLES_AUS,KANALWAHL+WRITE);
  Giaccess(ports,PORT_STATUS + WRITE);
}
```

Dieses Meeresrauschen hat nur einen einzigen Fehler, es verschwindet zeitweise ganz. Als Ausweg würde sich die Programmierung einer eigenen Lautstärkeregelung anbieten, die dann den Ausgangspegel zwischen beispielsweise 5 und 15 variiert.

Ein anderes Beispiel wäre die Synthese eines Schußgeräusches. Wenn wir die Explosion des Schwarzpulvers wahrnehmen, so wird die Lautstärke sicherlich bereits ihren maximalen Pegel erreicht haben, anschließend wird der Pegel recht schnell auf Null zurückgehen - schließlich ist die Kugel längst verschwunden und auch sonst ist nichts mehr zu hören. Diesem Verlauf der Lautstärkekurve entspricht nun exakt das Hüllkurvenmuster 4 aus unserer Übersicht:

```
/***************************************************************************/
/* sub5.c - PENG!                                                          */
/***************************************************************************/
sound()
{
   ports = Giaccess(dummy,PORT_STATUS);
   Giaccess(0,0 + WRITE);
   Giaccess(0,1 + WRITE);
   Giaccess(0,2 + WRITE);
   Giaccess(0,4 + WRITE);
   Giaccess(0,5 + WRITE);
   Giaccess(15,6 + WRITE);
   Giaccess(7,7 + WRITE);
   Giaccess(16,8 + WRITE);
   Giaccess(16,9 + WRITE);
   Giaccess(16,10 + WRITE);
   Giaccess(0,11 + WRITE);
   Giaccess(16,12 + WRITE);
   Giaccess(0,13 + WRITE);
   click();
   Giaccess(ALLES_AUS,KANALWAHL+WRITE);
   Giaccess(ports,PORT_STATUS + WRITE);
}
```

Damit soll es der Beispiele genug sein, denn als nächstes wollen wir die anderen Hüllkurven experimentell erproben. Dazu geben wir die Zeilen des Unterprogrammes TEST6.C ein, bei dem eine Betätigung der linken Maustaste jeweils die nächste Hüllkurve aktiviert:

```
/***************************************************************************/
/* sub6.c - Huelldemo                                                      */
/***************************************************************************/
sound()
{
int kurve;
   ports = Giaccess(dummy,PORT_STATUS);/* Portstatus auslesen           */
   kurve = 0;
   do{
      Giaccess(0,0 + WRITE);
      Giaccess(0,1 + WRITE);
      Giaccess(0,2 + WRITE);
      Giaccess(0,4 + WRITE);
      Giaccess(0,5 + WRITE);
      Giaccess(31,6 + WRITE);
      Giaccess(199,7 + WRITE);
      Giaccess(16,8 + WRITE);
      Giaccess(16,9 + WRITE);
      Giaccess(16,10 + WRITE);
      Giaccess(0,11 + WRITE);
      Giaccess(32,12 + WRITE);
      Giaccess(kurve,13 + WRITE);
      click();
      kurve++;
   }while (kurve<16);
   click();
   Giaccess(ALLES_AUS,KANALWAHL+WRITE);
   Giaccess(ports,PORT_STATUS + WRITE);
}
```

## 6.9 Tonprogramierung unter BASIC

Sie kennen nun die Fähigkeiten des in den ATARI ST eingebauten Soundchips und wissen diese zu nutzen. Doch werden die meisten von Ihnen, es sei denn, Sie wollen professionelle Spiele etc. in C programmieren, wohl eher den bequemen Weg über eine Interpretersprache vorziehen anstatt auf Bit- und Byteebene in den Eingeweiden des STs herumzuwühlen. Darum wird es Sie nun freuen zu hören, daß sämtliche Erklärungen und Beispiele des vorangegangenen Teiles übertragbar sind, und zwar sowohl für BASIC als auch für Logo. Es lassen sich weiterhin alle Generatoren ansteuern, allerdings wurde bei der Implementierung der Hochsprachen eher an den Musiker als an den Hardwarespezialisten und Computerfreak gedacht.

So kann es sich der Anwender, der die Tonerzeugung unter BASIC vornehmen möchte, beispielsweise ersparen, die Periode einer gewünschten Frequenz zu berechnen und anschließend auch noch in einen Low- und einen High- Wert zu zerlegen. Er muß sich nur folgende Zuordnung merken:




Damit sind die Noten einer Oktave verfügbar. Die Oktave selbst kann ebenfalls als Wert zwischen 1 und 8 angegeben werden. Alles was der Chip dann noch wissen muß, ehe er den gewünschten Ton spielen kann, ist der zu benutzende Tonkanal 1, 2 oder 3 und die Lautstärke (0 bis 15). Übermittelt werden ihm die Werte vereinbarungsgemäß in der Folge:

SOUND Kanal, Volume, Note, Oktave, Spieldauer

Um nun von BASIC aus den Kammerton A erklingen zu lassen, müssen wir also

SOUND 1,15,10,4,50

eingeben. Sicherlich werden auch Sie diesen Ton, der so lange erzeugt wird, bis Sie eine Taste drücken, als unseren altbekannten Testsound erkennen. (Vergleichen Sie dazu bitte auch die Tabelle im Anhang!)

Den gesamten Tonumfang führt Ihnen folgendes Programm vor:

```
10 REM Tonumfang
20 data 1,3,5,6,8,10,12,-1
30 for oktave = 1 to 8
40 restore
50 read note
60 sound 1,15,note,oktave,50
70 if note = 12 then 80 else 50
80 next periode
90 end
```

So können natürlich auch ganze Melodien gespielt werden, wie das Listing auf den folgenden Seiten zeigt.

```
10     fullw 2: clearw 2
20     print " **************  YOU CAN WIN IF YOU WANT  ***************"
30     read volume,note,oktave,dauer
40     if volume=-1 then end
50     sound 1,volume+4,note,oktave,dauer/3
60     i=i+1
70     if i<> 101 then 30
80     restore 280
90     read volume,note,oktave,dauer
100    if volume=-1 then end
110    sound 1,volume+4,note,oktave,dauer/3
120    i=i+1
```

```
130   if i<> 229 then 90
140   restore 420
150   read volume,note,oktave,dauer
160   if volume=-1 then end
170   sound 1,volume+4,note,oktave,dauer/3
180   i=i+1
190   goto 150
200   data 7,3,4,50, 7,3,5,100, 0,0,0,5, 7,3,5,50, 7,1,5,50
210   data 7,10,4,50, 7,8,4,25
220   data 7,10,4,25, 7,10,4,100, 7,3,5,50, 7,1,5,50
230   data 7,10,4,50, 7,8,4,50, 0,0,0,5, 7,8,4,50, 7,10,4,25
240   data 7,6,4,25, 7,6,4,75, 7,8,4,50, 7,10,4,25
250   data 7,6,4,25, 7,6,4,75, 7,8,4,50, 7,10,4,25
260   data 7,8,4,25, 7,8,4,50, 7,10,4,25, 7,1,5,25
270   data 0,0,0,5, 7,1,5,85
280   data 7,1,5,25, 7,3,5,25, 0,0,0,5, 7,3,5,50, 0,0,0,5
290   data 7,3,5,25, 0,0,0,5, 7,3,5,50, 0,0,0,5, 7,3,5,25
300   data 7,1,5,25, 7,5,5,25, 7,3,5,50, 7,1,5,25, 0,0,0,5
310   data 7,1,5,50, 7,10,4,25, 0,0,0,5, 7,10,4,25, 7,1,5,25
320   data 0,0,0,5, 7,1,5,25, 0,0,0,5, 7,1,5,25, 0,0,0,5
330   data 7,1,5,25, 0,0,0,5, 7,1,5,25, 7,10,4,50, 0,0,0,5
340   data 7,10,4,25, 0,0,0,5, 7,10,4,85, 0,0,0,5, 7,10,4,25
350   data 0,0,0,5, 7,10,4,25, 7,3,5,25, 0,0,0,5
360   data 7,3,5,25, 0,0,0,5, 7,3,5,25, 0,0,0,5, 7,3,5,25, 0,0,0,5
370   data 7,3,5,25, 7,12,4,75, 7,6,5,25, 0,0,0,5, 7,6,5,25
380   data 7,5,5,25, 7,3,5,25, 7,3,5,25, 0,0,0,5, 7,3,5,25, 7,1,5,25
390   data 7,3,5,25, 0,0,0,5, 7,3,5,25, 0,0,0,5, 7,3,5,25, 0,0,0,5
400   data 7,3,5,25, 7,3,5,25, 0,0,0,5, 7,5,5,25, 7,6,5,25, 7,3,5,25
410   data 0,0,0,5, 7,3,5,95,0,0,0,5
420   data 10,10,4,25, 10,1,5,25, 10,3,5,50, 10,3,5,25, 10,1,5,25
430   data 10,3,5,50, 10,4,5,25, 10,1,5,25, 10,3,5,25, 0,0,0,5
440   data 10,3,5,25, 0,0,0,5, 10,3,5,25, 10,1,5,25, 10,3,5,50
450   data 10,5,5,25, 10,6,5,25, 10,3,5,50, 0,0,0,5, 10,3,5,25
460   data 10,1,5,25, 10,3,5,25, 0,0,0,5, 10,3,5,25, 10,5,5,25
470   data 10,6,5,25, 10,8,5,25, 10,6,5,25, 0,0,0,5
480   data 10,6,5,25, 10,5,5,25, 10,3,5,50, 0,0,0,5
490   data 10,3,5,25, 10,1,5,25, 0,0,0,5, 10,1,5,50, 0,0,0,5
500   data 10,1,5,25, 10,12,4,25, 10,1,5,50, 10,8,4,25
510   data 10,12,4,25, 10,1,5,25, 0,0,0,5, 10,1,5,25, 0,0,0,5
520   data 10,1,5,25, 10,12,4,25, 10,1,5,50, 10,3,5,25, 10,5,5,25
```

```
530   data 10,3,5,75, 10,1,5,25, 10,10,4,120, 0,0,0,100
540   data 10,10,4,25, 10,1,5,25, 10,3,5,50, 0,0,0,5
550   data 10,3,5,25, 10,1,5,25, 10,3,5,50, 10,10,4,25
560   data 10,1,5,25, 10,3,5,25, 0,0,0,5, 10,3,5,25, 0,0,0,5
570   data 10,3,5,25, 10,1,5,25, 10,3,5,25, 10,10,4,50
580   data 10,11,4,25, 10,8,4,50, 0,0,0,5, 10,8,5,25, 10,6,4,25
590   data 10,8,4,50, 10,3,4,25, 10,6,4,25, 10,8,4,25, 0,0,0,5
600   data 10,8,4,25, 0,0,0,5, 10,8,4,25, 10,6,4,25, 10,8,4,50
610   data 10,3,5,25, 0,0,0,5, 10,3,5,25, 10,1,5,50, 0,0,0,5
620   data 10,1,5,25, 10,12,4,25, 10,1,5,50, 10,8,4,25
630   data 10,12,4,25, 10,1,5,25, 0,0,0,5, 10,1,5,25, 0,0,0,5
640   data 10,1,5,25, 10,12,4,25, 10,1,5,50, 10,3,5,25
650   data 10,5,5,25, 10,3,5,50, 10,1,5,50, 10,10,4,50
660   data 10,8,4,50, 10,10,4,25, 10,8,4,25, 10,6,4,50
670   data 0,0,0,50, 10,3,5,50
680   data 0,0,0,0, -1,-1,-1,-1
```

Doch wenn man weiß, was der Chip sonst noch kann, dann wird man sich nicht mit den reinen Tönen begnügen wollen. Zur Verfremdung kennt das ST BASIC die Anweisung WAVE. Die genaue Syntax lautet:

WAVE Enable,Lautstärkeregelung,Hüllkurve,Hüllk.periode,Dauer

Die einzelnen Parameter sind für uns längst alte Bekannte und bereiten uns keine Schwierigkeiten mehr. Einzig und allein die Parameter Lautstärkeregelung und Hüllkurvenperiode dürften eine kurze Bemerkung wert sein. Gemeint ist mit dem zweiten Wert hinter WAVE nämlich, ob der PSG die Lautstärke für einen bestimmten Kanal über eine Hüllkurve regeln oder stattdessen den in einer SOUND-Anweisung angegebenen Wert benutzen soll.

Beachten Sie den Unterschied: Hier legt ein einziges Datum die Inhalte der Register R8, R9 und R10 fest! Das heißt, daß hier irgendeine Codierung vorgenommen sein muß. Nun, die drei niederwertigsten Bits einer hier übermittelten Binärziffer sind von Bedeutung. Und zwar steht Bit 1 für Kanal 1 (A), Bit 2 für Kanal 2 (B) und Bit drei natürlich für Kanal 3 (C). Aktiviert

wird der entsprechende Hüllkurvengenerator, wenn das korrespondierende Bit gesetzt (=1) ist. Sind alle Bits gelöscht, wird also eine 0 übergeben, werden demgemäß alle drei Stimmen zusätzlich zur WAVE-Anweisung auch noch über ein SOUND-Kommando geregelt.

Das Setzen der Hüllkurvenperiode hat sich hingegen nicht verändert, sondern nur vereinfacht. Denn nun kann der 16-Bit-Wert auch als Zahl im Bereich zwischen 0 und 65535 übergeben werden; eine Aufteilung in LSB und MSB ist nicht mehr notwendig.

Wie die Beispiele auf der folgenden Seite zeigen, lassen sich allein mit diesen zwei Anweisungen, bei geschickter Parameterwahl, die gewünschten Effekte schnell und ohne große Rechenarbeit erreichen.

```
10    clearw 2: fullw 2
20    print "MINIORGEL"
30    print:print "Gespielt wird mit s,d,f,g,h,j,k."
40    print:print "Die Tonhoehe wird variiert mit + und -."
50    print: print "ENDE = CTRL-Z."
60    oktave=4
70    a$=input$(1)
80    if a$="s" then n=1 : goto 170
90    if a$="d" then n=3 : goto 170
100   if a$="f" then n=5 : goto 170
110   if a$="g" then n=6 : goto 170
120   if a$="h" then n=8 : goto 170
130   if a$="j" then n=10: goto 170
140   if a$="k" then n=12: goto 170
150   if a$="+" then oktave = oktave +1:if oktave >=8 then oktave =8
160   if a$="-" then oktave = oktave -1:if oktave <=1 then oktave = 1
170   sound 1,12,n,oktave,25
180   goto 70
190   for i=1 to 10:next:return
```

```
10   clearw 2: fullw 2 : z$=string$(60," ")
20   gotoxy 1,1: print " S O U N D E D I T O R":print
30   input "Oktave          ";oktave
40   input "Note            ";note : h=0
50   input "Spieldauer      ";dauer
60   input "Huellkurve (j/n) ";e$:if e$="j" then h=-1
70   if h=0 then enable=0:print z$:print z$:print z$:goto 110
80   input "Hüllkurve (0-15) ";kurve
90   input "Hüllperiode     ";hp
100  input "Enable (1-7)    ";enable
110  input "Kanaele (T/R)   ";kanal
120  wave kanal,enable,kurve,hp,dauer
130  goto 20
```

## 6.10 Der Soundchip unter Logo

Ähnlich verhält sich die Sache bei Dr. Logo. Auch hier läßt die Hochsprache dem Anwender wieder sämtliche Möglichkeiten von der Erzeugung eines simplen Tones bis zur Synthese komplexer Geräusche offen. Allerdings verhält sich die Sache längst nicht so komfortabel wie unter BASIC, arbeitet Logo doch wieder ausgesprochen registerorientiert - womit die Register R0 bis R13 des PSG gemeint sind!

Ein Beispiel sollte ausreichen, um Ihnen Syntax und Funktionsweise zu verdeutlichen:

```
TO MEER
SOUND [0 0 1 0 2 0 3 0 4 0 5 0 6 15 7 199 8 16 9 16 10 16 11
0 12 20 13 9]
END
```

Alles klar?

Nun, 14 Register können wir für die Soundsynthese nutzen - und die Zahlen 0 bis 13 tauchen auch in regelmäßigen Abständen innerhalb der Logo-Soundanweisung auf. Dazwischen steht jeweils eine andere Zahl: und genau diese wird in das zuvor genannte Register geschrieben!

Sie können sich bei der Soundprogrammierung unter Dr. Logo also durchweg an die Abbildung der Registerstruktur halten - wie wir es auch bei der Programmierung mit C getan haben. Das bedeutet natürlich auch, daß Sound-Effekte aus C-Listings problemlos übertragbar sind. Blättern Sie ruhig zurück und versuchen Sie es einmal.

Einige Anregungen finden Sie auch auf der folgenden Seite.

```
TO GUN
SOUND [0 0 1 0 2 0 3 0 4 0 5 0 6 15 7 7 8 16 9 16 10 16 11 0 12 16 13 0]
END

TO EXPLODE
SOUND [0 0 1 0 2 0 3 0 4 0 5 0 6 31 7 199 8 16 9 16 10 16 11 0 12 50 13
9]
END

TO MEER
SOUND [0 0 1 0 2 0 3 0 4 0 5 0 6 15 7 199 8 16 9 16 10 16 11 0 12 32 13
14]
END

TO PIANO
SOUND [0 200 1 0 2 201 3 0 4 100 5 0 6 0 7 248 8 16 9 16 10 0 11 0 12 20
13 9]
END

TO LOK
SOUND [0 0 1 0 2 0 3 0 4 0 5 0 6 31 7 199 8 16 9 16 10 16 11 0 12 6 13
12]
END
```

## 6.11 MIDI - was ist das?

Sofern Sie den vorangegangenen Teil dieses Kapitels aufmerksam durchgearbeitet haben, wissen Sie nun recht gut über die Fähigkeiten des ST in Bezug auf die Soundsynthese und das Spielen von Musikstücken Bescheid. Für einen Computeristen ist das ganze sicherlich recht eindrucksvoll und wer sich zum ersten Mal mit diesem Gebiet beschäftigt hat, der wird geradezu begeistert sein.

Doch ist dabei auch deutlich geworden, das der im ST eingebaute Tongenerator in erster Linie zur akustischen Untermalung von Telespielen gechaffen wurde. Dies zeigt sich zum Beispiel dadurch recht deutlich, das er in der Lage ist, unabhängig vom Prozessor zu agieren (beispielsweise mit den Hüllkurven 1110 und 1100), wie auch durch die nicht allzu komfortable Hüllkurvengeneration. So erlauben mehr auf Musikerzeugung getrimmte Soundchips unter anderem die Einstellung der Zeitdauer einer jeden einzelnen Hüllkurvenphase - und erst damit kann der Klang eines jeden Instrumentes perfekt nachgeahmt werden.

Und so wird ein (semi-)professioneller Musiker die Ergebnisse des dreistimmigen Soundchips eher belächeln. Allerdings ist der PSG für diesen ST-Anwender auch recht uninteressant, vielmehr wird das eingebaute MIDI-Interface seine Wertschätung finden. Kann er dank diesem doch mehrere Instrumente - zumindest soweit Sie neueren Datums und mikroprozessorgesteuert sind - gleichzeitig fernbedient spielen, und das ist nur ein Aspekt der weitreichenden Möglichkeiten des MIDI.

Technisch gesehen ist das 'Musical Instrument Digital Interface' nichts anderes als eine ganz normale serielle Schnittstelle - die sich von der 'normalen', die wir üblicherweise zur Daten(fern)übertragung per Akustikkoppler oder Nullmodemkabel benutzen nur durch ihre extrem hohe Übertragunsgeschwindigkeit unterscheidet.

Unzweifelhaft haben auch Sie schon des öfteren das Accessory RS-232-Einstellung angwählt, mit dessen Hilfe sich diese Schnittstelle einstellen läßt. Vorgesehen sind dort Übertragungs-

raten von 300 Baud bis 9800 Baud - wobei 1 Baud mit acht Bit pro Sekunde gleichzusetzen ist.

Wenn mann dann bedenkt, daß außerdem zur Unterscheidung der einzelnen Zeichen noch ein Start- und ein Stoppbit gesandt werden, so läßt sich absehen, daß auf diese Art und Weise zwischen 30 und 980 Zeichen pro Sekunde übertragen werden können.

Für Echtzeitanwendungen wäre das noch zu langsam, sollen alle Instrumente schließich im Takt spielen! Daher hat man sich bei der Festlegung der typischen Kenndaten dieser Schnittstelle, der 'MIDI-Spezifikation 1.0' aus dem Jahre 1982, auf eine Übertragungsrate von 31250 Baud geeinigt.

Vielleicht fragen Sie sich nun, warum mann dann nicht einer Parallelschnittstelle den Vorzug gegeben hat, können die acht Bits eines Bytes dann doch immer gleichzeitig statt nacheinander ausgegeben werden - was natürlich einen erheblichen Geschwindigkeitsvorteil bedeutet.

Doch treten bei diesem Verfahren Schwierigkeiten gleich in doppelter Hinsicht auf: denn erstens darf die Leitung dann kaum länger als drei Meter sein, und zweitens: stellen Sie sich einmal eine Bühne vor, auf der die Instrumente mit breiten (mindestens neunadrig!) und daher steifen Kabeln verbunden sind. Dort könnte sich kaum noch jemand bewegen ohne Gefahr zu laufen, alles umzureißen. Außerdem ist die Handhabung solcher Kabel sicherlich unbequemer als die einer dreiadrigen Leitung. Aus diesen Gründen gab man einer seriellen Schnittstelle, die sich beim MIDI immer als 5-polige DIN-Buchse präsentiert, den Vorzug.

Die technischen Daten dieses Anschlusses lesen sich folgendermaßen:

Übertragungsrate: 31250 baud
Datenbits: acht, zusätzlich 1 Start- und 1 Stoppbit
Übertragungsart: asynchron

Wobei asynchron nichts anderes bedeutet, als daß Daten nur bei Bedarf übertragen werden, es gibt also keine 'Nullbytes' die mit konstanter Regelmäßigkeit für eine Synchronisation der Geräte sorgen sollen.

Welcher Art sind nun die Daten, die das übertragen werden? Eigentlich sollte man meinen, daß diese Frage recht einfach zu beantworten ist, handelt es sich bei MIDI 1.0 doch um eine Normung. Doch weit gefehlt - zwar wurden die Übertragunsparameter festgelegt doch sind die benutzten Geräte relevant für die übertagenen Daten. Schließlich muß jeder Regler irgendwo im Übertragungsprotokoll auftauchen können. Für die wirklich in der Leitung übertragenen Signale verantwortlich ist jedoch das Mastergerät.

Womit sich die Frage aufwirft, was wir unter einem Master zu verstehen haben.

Nun, das Mastergerät ist das eigentliche Steuergerät, der Chef, der den übrigen an das MIDI-Netzwerk angeschlossenen Synthesizern, Rhythmusgeräten etc., den sogenannten Slaves die Bedienungsimpulse sendet.

Stellen wir uns vor, wir hätten zwei Synthesizer, beide natürlich mit einer MIDI-Schnittstelle ausgestattet, und unseren ATARI ST. Mit Sicherheit können wir dann an sämtlichen Geräten die beiden Buchsen MIDI IN und MIDI OUT entdecken, eventuell auch noch eine weitere Buchse mit der Bezeichnung MIDI-THROUGH.

Betrachten wir zunächst nur eine Verbindung der beiden Synthesizer: Bei dem einen Gerät stecken wir das Kabel in die MIDI OUT, und bei dem anderen in die MIDI IN-Buchse. Damit haben wir das erste Gerät zum Master erklärt, was nichts anderes bedeutet, als daß das zweite komplett vom ersten bedient werden kann. Natürlich nur unter dem Vorbehalt, daß dieses auch über genügend Regler verfügt, um alle Features des zweiten Gerätes nutzen zu können (umgekehrt können wir natürlich nicht erwarten, daß das zweite Gerät mehr kann bloß

weil das erste mehr Einstellmöglichkeiten bietet - und diese Einstellparameter ebenfalls per MIDI übertragen werden).

Hier wird deutlich, wie unterschiedlich die übertragenen Daten von Fall zu Fall sein können. Standardmäßig vorausgesetzt werden können eigentlich immer nur drei Übertragungsdaten:

- Note
- Anschlag
- Sound

Wobei letzterer schon wieder von den technischen Möglichkeiten des Gerätes abhängt.

Hinzu kommen optional Daten für:

- Synchronizität
- Reset
- Start/Stopsigale für Schlagzeuge usw.
- Lautstärke
- Herstellerdaten
- u.a.

Besonders die Herstellerdaten sorgen hier für voneinander verschiedene MIDI-Protokolle, so daß einem interessierten MIDI-Anwender nichts anderes übrigbleibt, als die 'system exclusiv-Daten' seines Gerätes genau zu studieren. Auch erlaubt MIDI in diesem Bereich das Ansprechen genau eines Gerätes. Dazu dient eine Adresse, die innerhalb des Musikinstrumentes per Schalter eingestellt werden kann (0 bis 15) - ein Verfahren, daß bereits seit langem den Anwendern von Commodore-Computern bekannt sein dürfte.

Nehmen wir den ST nun zur Gerätekonfiguration hinzu. Sobald wir die IN-Buchse eines Synthesizers an die OUT-Buchse des ST angeschlossen haben, ist der Computer der MASTER. Dann müßte sich der Synthesizer vollständig über den ST bedienen lassen können - selbstverständlich nicht, ohne geeignete Software.

Denn ehe der Synthesizer einen Ton erklingen läßt, erwartet er drei Bytes, nämlich:

1. ein Byte als Anweisung: Note spielen
2. ein Byte: welchen Ton (Note und Oktave)
3. ein Byte: Anschlag

Umgekehrt kann der ST (wiederum bei Verwendung geeigneter Software) auch als Tonbandmaschine benutzt werden. Dann wird er als Slave (also über MIDI IN) an die OUT-Buchse eines Synthesizers angeschlossen und die Töne werden in Echtzeit in den Computer überspielt. Dabei werden sämtliche Reglerstellungen des Synthesizers mitnotiert, so daß das Stück anschließend originalgetreu vom Computer gespielt werden kann!

Dies eröffnet natürlich phantastische Möglichkeiten für creative Musiker. Nicht nur, daß Routinearbeiten vereinfacht werden, denn selbstverständlich können so überspielte Melodien auch ausgedruckt werden. Vielmehr können diese Stücke (wie natürlich auch die Sounds) auf Diskette gespeichert und zu einem späteren Zeitpunkt editiert werden.

Und da MIDI das gleichzeitige Spielen von sechzehn computergesteuerten Musikintrumenten erlaubt, kann der ST sogar eine ganze Band ersetzen. Wenn nämlich ein Musiker nacheinander alle Daten eingespielt hat, und der ST als Master sämtliche Instrumente später absolut synchron und exakt spielt!

# Anhang

## A Programm zur Ermittlung der Periodenwerte des Tongenerators

```
10    ' Vergleichstabelle: Noten, Perioden & Frequenzen
20    '-----------------------------------------
30    clearw 2:fullw 2
40    dim zeichen$(12)
50    for note=1 to 12:read zeichen$(note):next
60    for oktave = -3 to 4
70    print:print "OKTAVE: ";oktave
80    print"Note            Frequenz        Periode"
90    for note = 1 to 12
100   frequenz = 440 * (2 ^ (oktave + ((note-10)/12)))
110   periode = int(125000 / frequenz)
120   print " ";zeichen$(note),frequenz,periode
130   next note
140   next oktave
150   data "C","C#","D","D#","E","F","F#","G","G#","A","A#","H"
```

## B Tabelle der Periodenwerte

OKTAVE: -3

| Note | Frequenz | Periode |
|---|---|---|
| C | 32,70321 | 3822 |
| C# | 34,64785 | 3607 |
| D | 36,70812 | 3405 |
| D# | 38,89089 | 3214 |
| E | 41,20347 | 3033 |
| F | 43,65355 | 2863 |
| F# | 46,24932 | 2702 |
| G | 48,99945 | 2551 |
| G# | 51,91311 | 2407 |
| A | 55 | 2272 |
| A# | 58,2705 | 2145 |
| H | 61,73544 | 2024 |

OKTAVE: -2

| Note | Frequenz | Periode |
|---|---|---|
| C | 65,40642 | 1911 |
| C# | 69,29567 | 1803 |
| D | 73,41621 | 1702 |
| D# | 77,78177 | 1607 |
| E | 82,40691 | 1516 |
| F | 87,30709 | 1431 |
| F# | 92,49863 | 1351 |
| G | 97,99889 | 1275 |
| G# | 103,8262 | 1203 |
| A | 110 | 1136 |
| A# | 116,541 | 1072 |
| H | 123,4709 | 1012 |

OKTAVE: -1

| Note | Frequenz | Periode |
|---|---|---|
| C | 130,8128 | 955 |
| C# | 138,5913 | 901 |
| D | 146,8324 | 851 |
| D# | 155,5635 | 803 |

| | | |
|---|---|---|
| E | 164,8138 | 758 |
| F | 174,6141 | 715 |
| F# | 184,9972 | 675 |
| G | 195,9978 | 637 |
| G# | 207,6524 | 601 |
| A | 220 | 568 |
| A# | 233,0819 | 536 |
| H | 246,9417 | 506 |

OKTAVE: 0

| Note | Frequenz | Periode |
|---|---|---|
| C | 261,6256 | 477 |
| C# | 277,1827 | 450 |
| D | 293,6648 | 425 |
| D# | 311,127 | 401 |
| E | 329,6276 | 379 |
| F | 349,2282 | 357 |
| F# | 369,9944 | 337 |
| G | 391,9954 | 318 |
| G# | 415,3047 | 300 |
| A | 440 | 284 |
| A# | 466,1638 | 268 |
| H | 493,8833 | 253 |

OKTAVE: 1

| Note | Frequenz | Periode |
|---|---|---|
| C | 523,2511 | 238 |
| C# | 554,3653 | 225 |
| D | 587,3295 | 212 |
| D# | 622,254 | 200 |
| E | 659,2551 | 189 |
| F | 698,4565 | 178 |
| F# | 739,9888 | 168 |
| G | 783,9908 | 159 |
| G# | 830,6093 | 150 |
| A | 880 | 142 |
| A# | 932,3273 | 134 |
| H | 987,7664 | 126 |

OKTAVE: 2

| Note | Frequenz | Periode |
|---|---|---|
| C | 1046,502 | 119 |
| C# | 1108,73 | 112 |
| D | 1174,659 | 106 |
| D# | 1244,508 | 100 |
| E | 1318,51 | 94 |
| F | 1396,913 | 89 |
| F# | 1479,977 | 84 |
| G | 1567,981 | 79 |
| G# | 1661,218 | 75 |
| A | 1760 | 71 |
| A# | 1864,654 | 67 |
| H | 1975,533 | 63 |

OKTAVE: 3

| Note | Frequenz | Periode |
|---|---|---|
| C | 2093,004 | 59 |
| C# | 2217,46 | 56 |
| D | 2349,318 | 53 |
| D# | 2489,015 | 50 |
| E | 2637,02 | 47 |
| F | 2793,825 | 44 |
| F# | 2959,954 | 42 |
| G | 3135,962 | 39 |
| G# | 3322,436 | 37 |
| A | 3520 | 35 |
| A# | 3729,309 | 33 |
| H | 3951,066 | 31 |

OKTAVE: 4

| Note | Frequenz | Periode |
|---|---|---|
| C | 4186,008 | 29 |
| C# | 4434,92 | 28 |
| D | 4698,634 | 26 |
| D# | 4978,03 | 25 |
| E | 5274,038 | 23 |
| F | 5587,649 | 22 |
| F# | 5919,909 | 21 |
| G | 6271,924 | 19 |

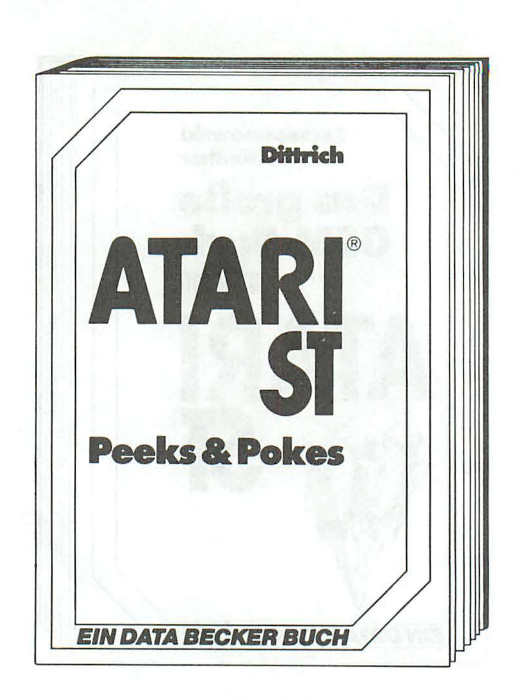
| | | |
|---|---|---|
| G# | 6644,871 | 18 |
| A | 7040 | 17 |
| A# | 7458,616 | 16 |
| H | 7902,127 | 15 |




Den ATARI ST voll ausnutzen können Sie nur in Maschinensprache! Zahlensysteme, Bitmanipulation, der 68000 im ATARI ST, Registerverwendung, Struktur des Befehlssatzes, Programmstrukturen, Rekursion, Stacks, Prozeduren, Grundlagen der Assemblerprogrammierung Schritt für Schritt, Verwendung von Systemroutinen und Tips zum Einbinden von Assemblerroutinen in Hochsprachen. Eine hervorragend geschriebene Einführung!

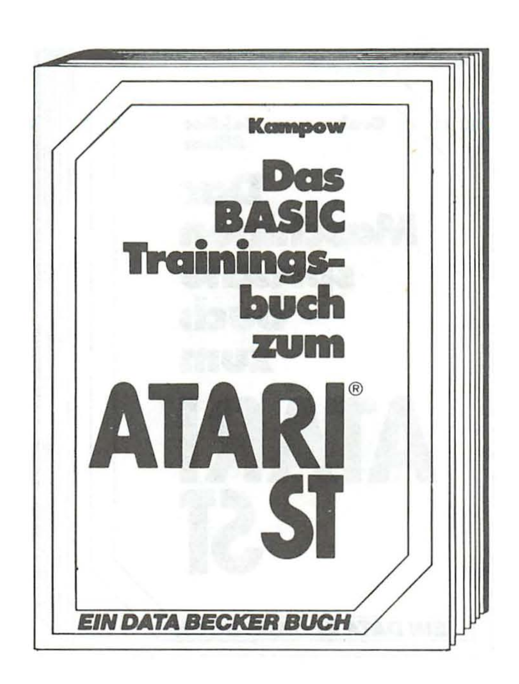
**Grohmann/Slibar/Seidler**
**ATARI ST Maschinensprache**
**250 Seiten, DM 39,–**
**ISBN 3-89011-120-3**


Sauer
Das LOGO Trainingsbuch zum
ATARI ST
EIN DATA BECKER BUCH




Schlagen Sie dem Betriebssystem Ihres ATARI ST ein Schnippchen. Wie? Mit PEEKS & POKES natürlich! Dieses Buch erklärt Ihnen leichtverständlichh den Umgang damit. Mit einer riesigen Anzahl wichtiger POKES und ihren Anwendungsmöglichkeiten. Dabei wird der Aufbau Ihres ST's prima erklärt: Betriebssystem, Interpreter, Zeropage, Pointer und Stacks sind nur einige Stichworte dazu. Der erste Schritt hin zur Maschinensprache!

**PEEKS & POKES zum ATARI ST**
**194 Seiten, DM 29,–**
**ISBN 3-89011-148-3**


Kampow
Das BASIC Trainingsbuch zum
ATARI®
ST
EIN DATA BECKER BUCH

***DAS STEHT DRIN:***

Lernen Sie die ungewöhnlichen Grafik- und Soundfähigkeiten des ATARI ST kennen – mit diesem Buch, das Ihnen neben einem guten Einstieg viele Beispiele und sinnvolle Hilfsprogramme bietet. Dabei werden GEM und BASIC, Logo, C und auch Modula 2 berücksichtigt, so daß mit Sicherheit für jeden etwas dabei ist.

Aus dem Inhalt:

- Raster- und Vektorgrafik
- Spiegelungen und Rotationen
- 2- und 3-D-Funktionsplotter
- Grafik unter GEM
- Einführung in die Grafik mit Logo
- Moiré-Figuren, Kreis- und Balkendiagramme
- Shapes
- Koordinatentransformationen
- Grundlagen der synthetischen Musik
- Lautstärke-Hüllkurven
- Der Sound-Chip
- Sound-Chip unter Logo
- Der ATARI als Synthesizer

***UND GESCHRIEBEN HAT DIESES BUCH:***

Jörg Walkowiak ist Informatik-Student und Fachautor zahlreicher DATA BECKER-Bücher, mit besonderem Faible für Grafikprogrammierung.

***ISBN 3-89011-123-8***